AN ALGORITHM FOR INTRINSIC GAIN AND OFFSET STABILIZATION OF PULSE HEIGHT SPECTRA

by

Jiang-hong Lu

Submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January, 1991

Signature of Author _____
Department of Electrical Engineering and Computer Science
January 15, 1991

Certified by _____
Professor George C. Verghese
Thesis Supervisor (Academic)

Certified by _____ January 11, 1991
Dr. Charles C. Watson
Thesis Supervisor (Schlumberger-Doll Research)

Accepted by _____
Professor Arthur C. Smith
Chairman, Department Committee on Graduate Students

Table of Contents

Dedication

To My Parents, Grandpa and Jiqing

with Love

Acknowledgement

I am very grateful to my thesis supervisor, Professor George Verghese, for his guidance and encouragement.

I am very indebted to my thesis supervisor, Dr. Charles Watson, who supervised me through out the thesis project. I admire his enthusiasm in research and his meticulous scholarship. I have learned a tremendous amount from Charles during my internship at Schlumberger-Doll Research.

I thank Tarek Habashy and Joel Groves for their guidance and support for my work at Schlumberger-Doll Research.

I thank Darwin Ellis, Joseph Chiaramonte, James Grau, David Rossi, Raymond Kocian, Dianne Hughes and Sheila Monheit for helping me with the computer work.

I thank Joseph Doucet, Felix Chen, Peggy Dow, Qing-huo Liu, Christian Straley, Yupai Hsu and all other friends at Schlumberger-Doll Research for their friendship and support.

AN ALGORITHM FOR INTRINSIC GAIN AND OFFSET STABILIZATION OF PULSE HEIGHT SPECTRA

by

Jiang-hong Lu

Submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

on January 15, 1991 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in Electrical Engineering and Computer Science

Abstract

Multichannel pulse height spectra (PHS) are often used as property measurements of rock formations in well-logging. Gain and offset errors occur due to the many factors, and need to be corrected. The conventional method is to use stabilization source for calibration. No method of intrinsic gain and offset correction has been reported. An estimation algorithm is developed for intrinsic gain and offset stabilization.

Results from testing the estimation algorithms showed the estimated gain and offset fit the actual gain and offset well.

Thesis Supervisor: Dr. Charles C. Waston
Title: Research Scientist, Schlumberger-Doll Research

Thesis Supervisor: Dr. George C. Verghese
Title: Associate Professor of Electrical Engineering

Chapter 1 Introduction

Gamma ray pulse height spectra (PHS)[1] have been used for a long time in radiation detection, such as underground nuclear bomb detections in the 1940's, and environmental radiation measurements [8] in the past thirty years. In well logging, pulse height spectra obtained from multiply scattered gamma-ray devices are used for measurements of rock formation properties[3]. A typical gamma ray device consists of a radioactive source emitting gamma rays[2] into the rock, and a scintillation detector recording the numbers of accumulated photons at certain energy levels from the emitted gamma rays which are scattered back from the rock. In the measuring process, a gamma ray device is put in a well against the rock formation and then moved up smoothly along the formation. Over every small distance (usually 1/4 inch) scanned by the device, a pulse height spectrum is accumulated. Some of the spectra are shown in Figure 1.1. The interaction between the photon flux of gamma rays and the rock formation is primarily determined by the rock's electron density and the Compton and photoelectric cross sections. Therefore, the pulse height spectra accumulated from a rock formation can be used to determine the rock's density and effective atomic number.

Gamma ray pulse height spectra are measured by scintillation counters. A scintillation detector records detected photons into different energy channels which forms the multichannel pulse height spectra. The photon energy and the channel number have the following approximately linear relationship:

$$E \approx G\,C + O$$

where E is the energy deposited by the photons in the scintillation counter, G is energy gain, C is channel number, and O is energy offset. Gain and offset are parameters related to the scintillation counter. Gain is the energy variation range per channel, and offset is the background energy due to the DC voltage level applied to the scintillation detector.

---

[1] Measured by scintillation detectors. See Section 2 for detail.

[2] Gamma rays are formed by high energy photons.

During a measurement, both gain and offset could vary due to changes in many factors. The temperature of the device is often a very significant factor. Other factors can be the electronic circuitry due to the aging of the device, the electromagnetic field in the device, mechanical shocks and so forth. The relationship between the gain and offset variations and the changes in the influencing factors is generally complex and unknown. Gain and offset variations in the detector system are typically more significant in well logging than in the laboratory measurements. Corrections for gain and offset are essential before the pulse height spectra can be made useful.

Over the years, people in laboratories and well logging fields have developed many different ways for gain and offset corrections during the measuring process. The conventional method is to use a gain calibration source, but this has some unavoidable shortcomings discussed in Section 2.

In this study we will attempt to develop an alternative procedure for the gain and offset stabilization of the multiply scattered gamma ray devices used in borehole logging. From Dr. Watson's research [13], we know that even though a gamma ray pulse height spectrum's shape (see Figure 1.1) depends on the physics properties of the rock formation, every spectrum can be expressed as the sum of five or six basis spectra[3]. These basis spectra are associated with each device at a certain gain and offset. If a spectrum measured by the same device cannot be reconstructed well by the basis spectra, the spectrum must have different gain and offset values than the values associated with the basis spectra. We already have a Fortran subroutine which adjusts a spectrum with different gain and offset values. To make gain and offset corrections of a spectrum, we can use the subroutine to adjust the spectrum with different combinations of gain and offset until we find the gain and offset values which give the best basis spectra reconstruction of the spectrum. This gain and offset are then our estimated values.

Our algorithm which does not rely on a gain stabilization source is an alternative to the conventional technique. Our goal is to implement the algorithm and test its performance. In Chapter 2, we discuss the background of gain and offset variation problems, the conventional method for making gain and offset corrections, and the structure of tools

---

[3] See Section 3 for details

used in well logging. In Chapter 3, we analyze the characteristics of the pulse height spectra and present our approach to gain and offset adjustment. In Chapter 4, we address our estimation algorithm for gain and offset. In Chapter 5, we show the performance of our estimation algorithm. Finally, in Chapter 6 we draw conclusions on the research.

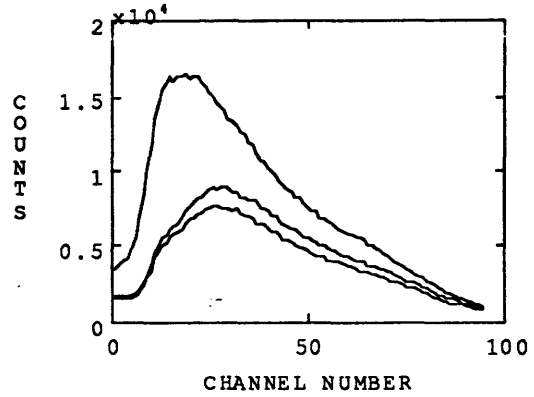

Figure 1.1: Pulse Height Spectra obtained from multiply scattered gamma ray device

Chapter 2. Problem Background

In this chapter, we examine the structure of scintillation counters and gamma-ray devices used in borehole logging and discuss the factors that might cause variations in gain and offset. We then study the conventional method for gain and offset determination and explain why more work needs to be done on gain and offset stabilization.

2.1 Scintillator Counter

The detection of ionizing radiation by scintillation light produced in certain materials is one of the oldest radiation detection techniques on record. The scintillation process remains one of the most useful methods available for the detection and spectroscopy of a wide assortment of radiations [6]. A typical scintillation counter is illustrated schematically in Figure 2.1. The device consists of a crystal scintillator detector, a photomultiplier tube and a circuit for photon energy resolution.

A crystal scintillator detector converts high energy photons into low energy light photons. Usually, photons emitted from a radiation process, such as gamma rays, possess very high energies. When a high energy photon enters a crystal scintillator detector, the photon interacts with an electron inside the crystal and transfers part of its energy to the electron, which excites the electron to a new energy state in which the electron is called a *secondary* electron. The photon then interacts with another electron, loses more energy to excite the electron. The process goes on until the photon is absorbed or escapes from the crystal. The secondary electrons release the absorbed energy by emitting photons which are usually light photons with relatively low energies. Typically, a high energy photon is converted into a few hundreds light photons.

A scintillator crystal should convert the kinetic energy of charged particles into detectable light with a high scintillation efficiency. For absolute measurements of photon energy, it is most convenient if the conversion is linear -- the light yield should be proportional to deposited energy over as wide a range as possible. The discovery of thallium activated sodium iodide in the early 1950s began the age of modern spectroscopy

of gamma rays. The most notable property of NaI(T1) is its excellent light yield, which is among the highest of any known scintillation material to secondary electrons. Its response to electrons (and any gamma rays) is close to linear over most of the significant energy range which includes the energy band from 10 keV to 700 keV that we are interested in.

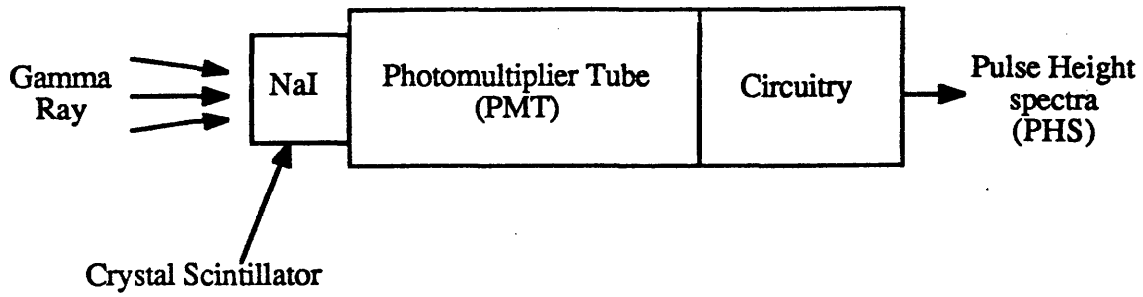

Figure 2.1: Schematic of a scintillation counter

The photomultiplier tube (PMT) converts light signals that typically consist of no more than a few hundred photons into a usable current pulse which is recorded by the circuits. The conversion is accomplished in three steps which are similar to the conversion in a crystal scintillator. First, a light photon from the crystal scintillator is absorbed at the photocathode raising an electron to an excited state. Then the electron escapes from the photocathode through photoelectric emission and enters the electron multiplier where the energy deposited by the electron results in the emission of more electrons producing an avalanche. Finally, the secondary electrons form a electric pulse at the collector,.which is recorded into a particular channel of a multichannel analyzer according to its amplitude (pulse height). Some of the pulse height spectra are shown in Figure 1.1.

Due to the approximately linear conversion in both the scintillator and the photomultiplier tube, the relationship between the energy deposited by a gamma ray and the recording channel number is also approximately linear [5]:

$$E \approx G\,C + O \qquad < 2.1 >$$

where E represents the energy deposited by the high energy photon (keV), G represents energy gain (keV / channel), C represents channel number and O represents offset (keV).

2.2 Multiply Scattered Gamma-Ray Devices

In well logging, the interaction between gamma rays and the rock formation is measured to determine the electron density and the average photoelectric cross section of the rock. Figure 2.2 shows a schematic of a typical multiply scattered gamma ray device used in laboratories, which is formed by a radioactive source emitting 662 keV gamma rays into the rock, and a scintillation counter detecting the gamma rays scattered back at energies greater than 10 keV. In between the radioactive source and the scintillator detector is shielding which prevents gamma rays from directly entering the scintillator without travelling through the formation. Near the detector are two optional stabilization sources for gain and offset calibration[4]. As the device moves along the formation, a pulse height spectrum is obtained for every spatial interval by integrating over a uniform time interval. Examples of typical spectra are shown in Figure 1.1. Typically, the integrating spatial interval is about a quarter of an inch, and the integrating time interval is no greater than one second. In a multichannel pulse height spectrum, every channel contains the number of photons (counts) at a certain energy level. If we divide the counts by its integrating time interval, we get the *count rate* (counts / second) which will be used in the rest of our analysis.

A multiply scattered gamma ray device used in well logging is moved up through a borehole against the rock formation to make measurements. *Logging speed* (inch / second) describes how fast the device moves in a borehole during measurements. The device is usually similar to the one shown in Figure 2.2.

---

[4] See Section 2.4 for details.

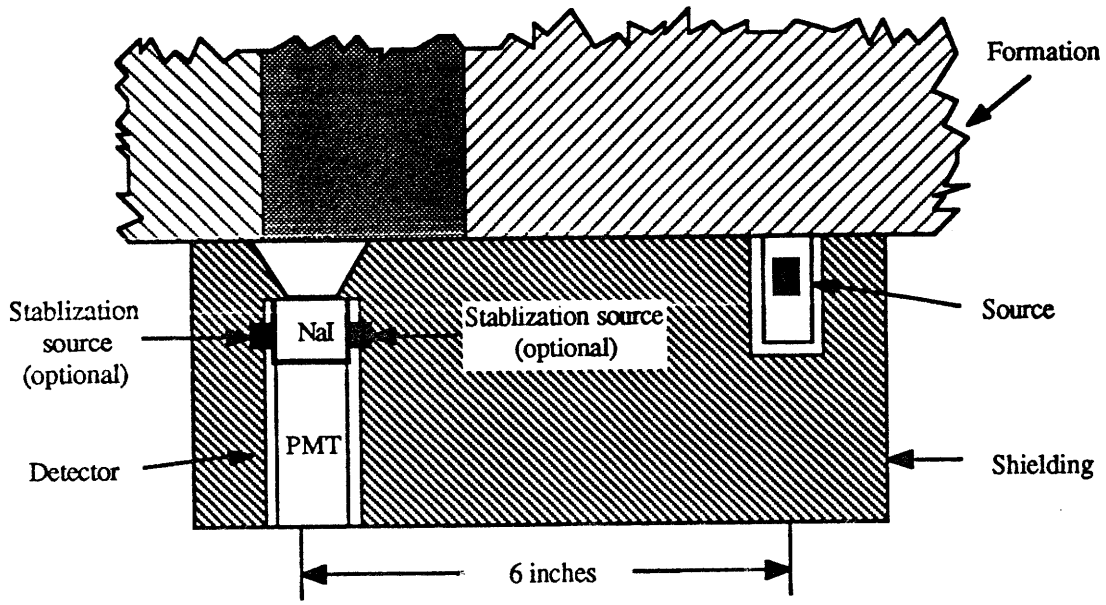

Figure 2.2: Schematic of a multiply scattered gamma ray device

2.3 Gain and Offset Variations

Many factors, especially temperature of a scintillator detector, count rate and mechanical shocks, can cause gain and offset to vary. Gain changes are dominated by changes in the photomultiplier tube. One very important type of gain change is the dynamic change due to large variations in the throughput of the photomultiplier, for the photomultiplier is not necessarily a linear amplifier. Count rate is determined by the throughput in the scintillator. The precise relationship between gain and count rate is usually unknown. Offset changes are primarily due to changes in the DC voltage level ("ground" potential) in various parts of the circuit. There can also a contribution from "dark current" at high temperatures (eg. 150° C). A DC voltage applied to a photo multiplier tube or a capacitor can cause a current ("dark current") when the temperature of the device is high enough. Offset changes are usually not as serious as gain changes. In well logging, it is common to make corrections only for gain.

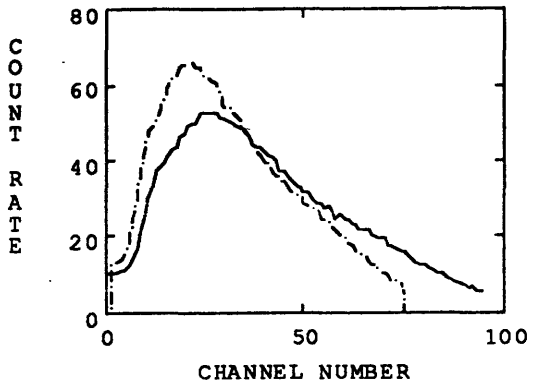

Figure 2.3: Effect of gain change on a spectrum
The offset for both spectra is 0.0 keV. The gain for the original spectra (solid curve) is 3.2 keV/channel. The dashed curve is the equivalent spectrum of the original when gain is 4.0 keV/channel.

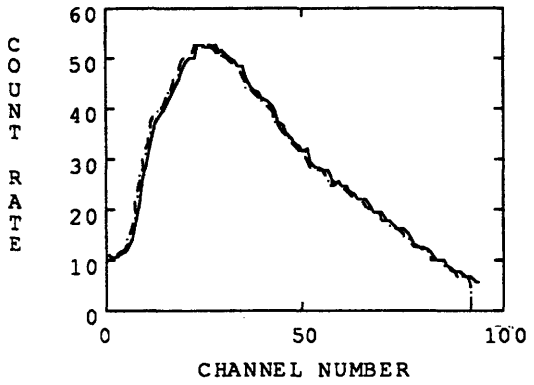

Figure 2.4: Effect of offset change on a spectrum
The gain for both spectra is 3.2 keV/channel. The offset for the original spectra (solid curve) is 0.0 keV. The dashed curve is the equivalent spectrum of the original when offset is 3.2 keV.

For a device, the gain and offset could vary significantly during a measurement either in a laboratory or a logging well. A one hundred percent change in gain or offset is quite common in laboratories. In well logging, the variations are much greater. As we know from Equation < 2.1 >, the apparent energy deposited by a high energy photon into

the detector is determined by the gain and offset of the device and the channel number from the measurement. If the gain and offset are wrong, the energy of the detected particles would be misinterpreted. Therefore, it is imperative to recognize changes in gain and offset for any measurement. A change in gain either "squeezes" or "stretches" the spectrum and a change in offset simply shifts the spectrum. Effects of gain and offset changes on a spectrum are shown in Figure 2.3 and Figure 2.4.

2.4 Conventional Measurements for Gain and Offset

For any precise measurements, the detector system is calibrated by measuring the gain and offset of the device. A technique of using one or more stabilization sources to do the calibration is very effective and widely used in laboratory measurements and well logging.

In the laboratory, radioactive materials of known radiation energies are used as stabilization sources. For example, cesium (^{137}Cs) emits 662 keV gamma rays, and americium (^{241}Am) emits 60 keV gamma rays. When measuring gain and offset of the detector on a multiply scattered gamma ray device, put the stabilization sources cesium and americium in front of the scintillator detector and get a pulse height spectrum. A good measurement should result in a calibration spectrum similar to the one shown in Figure 2.5. The spectrum should have two clear peaks around 60 keV (E_1) and 662 keV (E_2). From the spectrum, we know the channel numbers C_1 and C_2 where the peaks occur. The relationship between the energy E and channel number C is not absolutely linear. However, the non-linearity is not important for well logging applications. Empirically, two point calibration is adequate for gain and offset using Equation < 2.1 >:

$$E_1 \approx G\,C_1 + O \qquad < 2.2 >$$

$$E_2 \approx G\,C_2 + O \qquad < 2.3 >$$

If offset is under control and only gain needs to be measured, either one of the two peak points can be used.

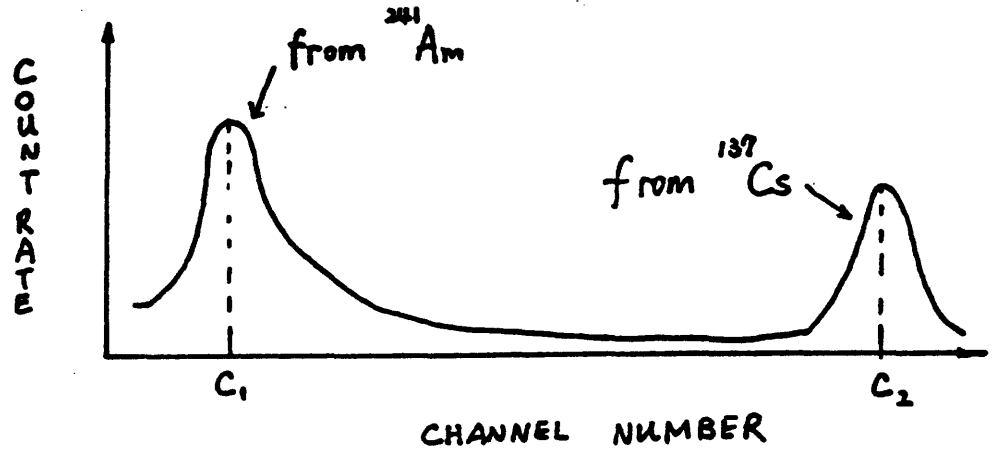

Figure 2.5: A calibration spectrum

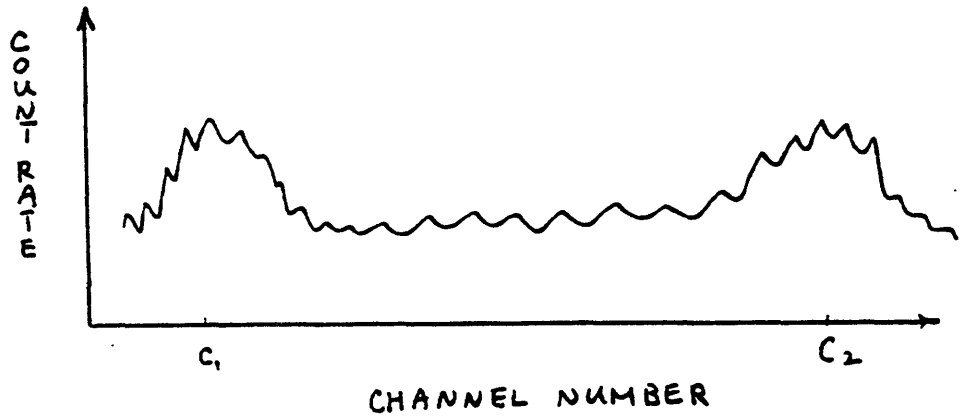

Figure 2.6: Poorly measured calibration spectrum

Very often, poor measurement results as shown in Figure 2.6 occur due to inadequate statistics. The results can be improved by using longer integrating time intervals in order to record more photons which might make the peaks stand out more. In well logging, however, the logging speed[5] cannot be reduced indefinitely for both technical and economical reasons. Moreover, an increase in integrating time interval causes a reduction in

[5] See Section 2.2 for definition.

spatial resolution. Also some of the photons from the source deposit only part of their energy and escape from the detector, which causes background noise in the low energy levels of the spectrum.

Gain and offset corrections are usually straightforward in the laboratories, because conditions can be controlled so that gain and offset changes occur slowly compared to the time duration of a measurement. In the laboratory measurements, the device can be moved very slowly so that an increase in integrating time interval will not reduce spatial resolution. Therefore, the laboratory data have better statistics.

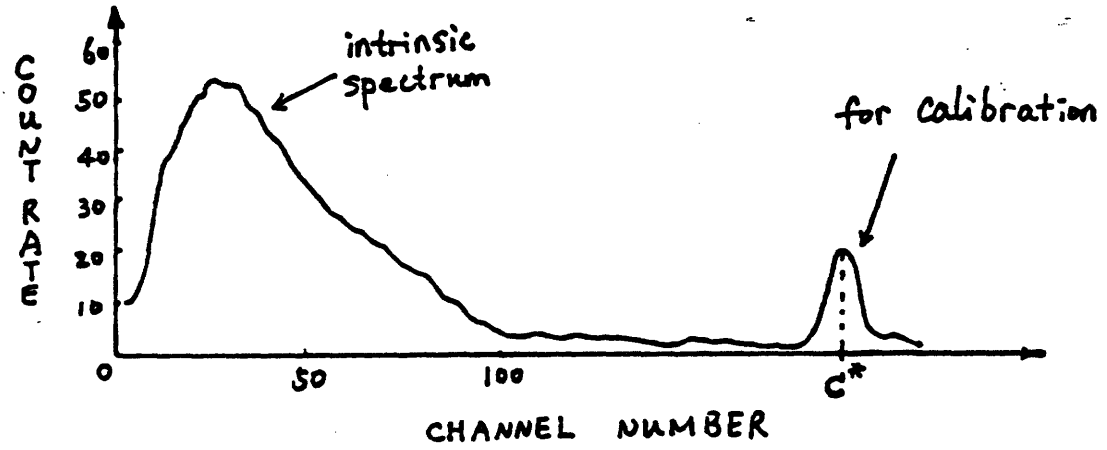

Figure 2.7: A PHS with a cesium stabilization source

In the fields of well logging, the same technique is applied for gain and offset calibration. As shown in Figure 2.2, one or more stabilization sources are placed at the scintillator detector. If cesium is used as both the device source and the stabilization source, the measured spectrum should be similar to the one shown in Figure 2.7. Gamma rays from the stabilization source enter the detector directly and form a peak in the spectrum around 662 keV, which can be used for gain calibration. Gamma rays from the device source have to travel through the rock formation and part of their energies are lost in the formation. Therefore, the gamma rays from the device source form the lower energy part of the spectrum, approximately from 10 keV to 500 keV. Usually, proper hardware design

can reduced offset variations so that an offset calibration measurement is not essential. On the other hand, gain is much more difficult to control. Typically in well logging, only a gain correction measurement is performed.

In well logging, measurements obtained from multiply scattered gamma ray devices by using stabilization sources cannot perform as well as in the laboratories for the following reasons.

1. The tool is often operated at the ambient borehole temperature which varies with depth and may go up to 150° C or higher. The wild temperature change causes both gain and offset to vary significantly and makes it hard to accomplish gain and offset calibration.
2. Measurements in well logging are made very rapidly compared to the laboratory measurements, which results in poor statistics in gain calibration peak.
3. The crystal scintillator is usually small, thus the background noise at low energy levels from the stabilization source is large, and the sensitivity of the measurement is degraded.
4. The tool is being moved up or down the borehole and is subject to mechanical shocks. Fast and dynamic changes in the measurements are hard to handle with the stabilization source technique.

We are motivated by the need to improve the stabilization of gamma ray spectral measurements under well logging conditions. In our research, we try to develop another technique for gain and offset stabilization without using the stabilization sources.

Chapter 3. Spectral Analysis and Numerical Approach

We mentioned in the Introduction that any pulse height spectrum obtained from a multiply scattered gamma ray device (see figure 2.2) can be expressed as the sum of a set of basis spectra with the same gain and offset as the measured spectrum. The shape of a PHS can vary significantly due to the physical variations in density and effective photoelectric factor [6] P_e of the measured rock formation. But as long as the gain and offset do not change, the PHS can be well represented in terms of the same set of basis spectra.

Gain and offset changes affect the "shape" of a pulse height spectrum, which usually results in a dilation and translation of the spectrum (see figure 2.3 and 2.4). A pulse height spectrum with gain G and offset O can be mapped into the equivalent spectrum with gain G' and offset O' which are different from G and O. The mapping can be done with some subroutine[7]. We hope that the changes in gain and offset cause enough change in the shapes of PHS to be recognized.

We want to find a set of basis spectra for a multiply scattered gamma ray device with known gain and offset. Then with the help of the mapping subroutine, we can estimate the gain and offset of a PHS by mapping it from possible gains and offsets into the gain and offset of the basis spectra and then reconstructing the PHS with the basis spectra. The best reconstruction should occur when the PHS is mapped from its true values of gain and offset.

3.1 The Data

We use lab data for algorithm development and performance testing. Our algorithm is intended to improve the gain and offset stabilization for field data obtained from borehole logging. However, field data have unknown changes in gain and offset, which make it

---

[6] By convention, the photoelectric factor of a material is defined as $P_e = (Z / 10)^{3.6}$, where Z is the atomic number.

[7] In implementation, we tried three mapping subroutines which are discussed in Chapter 4.

impossible to test the performance of our algorithm. Field data also consists of a higher level of counting noise. The gain and offset for lab data are known, or the changes in gain and offset are small. Therefore, we can map the lab data into other gain and offset values or add counting noise (usually Poisson noise), and then use the original data for calibration.

Our analysis is based on the ensemble of pulse height spectra acquired by Dr. Arthur Becker in the laboratory at Schlumberger-Doll Research. The type of device shown in Figure 2.2 were used to scan a synthesized rocklike formation to obtain the PHS. In Figure 2.2, the spacing between the source and the scintillator detector is 6 inches. The three scans in the ensemble used in our analysis are all obtained from the device shown in Figure 2.2.

The laminated formation used for data acquisition was synthesized from 35 sheets of 9 different rock or rocklike materials varying in density from 1.8 to 3.0 g/cc and in the effective averaged photoelectric factor[8], P_e, from 0.2 to 6. These materials were randomly layered transversely to the scan direction. PHS were accumulated at 0.25 inch intervals along the face of this formation in order to adequately sample possible spectral variations due to property transitions. In the analysis, 240 spectra per scan are used and each spectrum consists of 94 channels[9]. In Figure 2.2, the scan was made with the device immediately adjacent to the face of the formation, which is called *no stand-off*. Two other scans were made with the same device and formation by inserting thin sheets of different materials between the device and the face. The thin sheets were 0.21 inch thick plexiglass and 0.26 inch thick barite rubber. The three scans form the ensemble of measurements for the device. The ensemble has uniform gain and offset of 3.2 keV/Channel and 0.0 keV respectively. Table 1 shows the file names of the data used in our analysis.

The ensemble of measurements give fairly complete coverage of spatial variations in the direction of the scan, but limited variations in the directions orthogonal to the scan

---

[8] By convention, the photoelectric factor of a material is defined as $P_e = (Z / 10)^{3.6}$, where Z is the atomic number.

[9] The original PHS acquired by Dr. A. Becker consists of 99 channels in every spectrum. The first five channels in most of the spectra are zeros. Therefore, we only include the last 94 channels for each spectrum in our data files.

direction. In our study, we will use all scans in the the ensemble. In Chapter 5, we show some analysis on scan sd133, sd133a in Table 1.

Table 1 Scan File Names

Materials inserted	Scan File Name
No Stand-off	sd133
plexiglass (0.21 in.)	sd133a
barite rubber (0.26 in.)	sd133c

3.2 Principal Components

From the previous discussion, we want to find a set of basis spectra for a multiply scattered gamma ray device at certain gain and offset. One such set can be formed by the principal components which were discussed in Dr. Watson's paper [13].

First, we define each spectrum from an ensemble as a column vector **z** of dimension n.[10] Then the spectral covariance matrix is formed by averaging over the scans:

$$R_{zz} \equiv \langle (z - \langle z \rangle)(z - \langle z \rangle)^T \rangle = U M U^T \qquad < 3.1 >$$

where the brackets $\langle\rangle$ signify an ensemble average. Because the covariance matrix R_{zz} is real and symmetric, it can be diagonalized as indicated above. The matrix U is orthogonal in which each column, u_i, is an eigenvector of R_{zz}. These eigenvectors are also known as the *principal components* of the ensemble. The plot of the first four principal components of scan sd133 are in Figure 3.1. The matrix M is diagonal and contains the eigenvalues of R_{zz}, $\{\mu_i\}$. These eigenvalues are also the variances of the principal components and thus all the μ_i's are non-negative. By convention, the μ_i's are ordered by descending magnitude. The plot of the variances is in Figure 3.2.

---

[10] The number of channels in a spectrum is n.

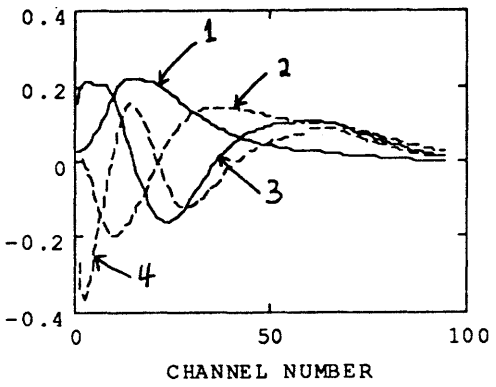

Figure 3.1: The first four principal components from scan sd133

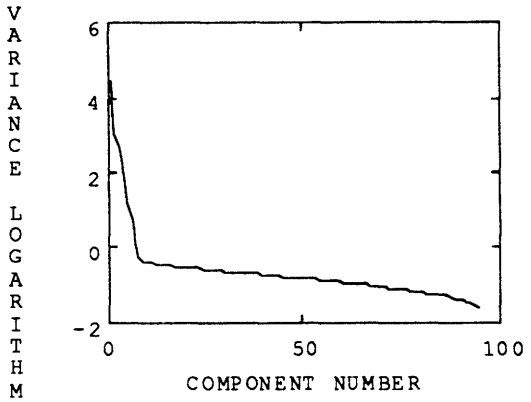

Figure 3.2: The logarithm variance of scan sd133

The magnitude of spectral variances decrease rather quickly over the first few components and the magnitude of the rest of the components are negligible. Let **z** be a spectrum which has the same variance for the principal components. If we use just the first *m* principal components to estimate the spectrum **z** , we get

$$\hat{z}_m = \sum_{i=1}^{m} a_i u_i + \langle z \rangle \qquad < 3.2 >$$

and the *mean* square error:

$$\varepsilon^2(\hat{z}_m) = \left\langle (\hat{z}_m - z)^T (\hat{z}_m - z) \right\rangle \qquad < 3.3 >$$

is minimized at :

$$a_i = u_i^T (z - \langle z \rangle). \qquad < 3.4 >$$

where a_i is obtained by projecting vector z on principal component u_i. The minimum square error is then:

$$\varepsilon^2(\hat{z}_m) = \left\langle (\hat{z}_m - z)^T (\hat{z}_m - z) \right\rangle = \sum_{i=m+1}^{n} \mu_i$$

The mean square error is negligible when m is between 4 and 6 because the first few principal components contain almost all the variance of z. Figure 3.3 and Figure 3.4 show two spectra from scan sd133 and their reconstructions with four principal components computed from scan sd133. From the two plots, we see that the reconstruction errors are very small even though the two spectra were measured from formations with different densities.

The above phenomena can be interpreted as follows. The first few principal components contain the independent modes of variation in PHS while the rest of the principal components contain mostly noise. Therefore, we can use the first few principal components to form the set of basis spectra.

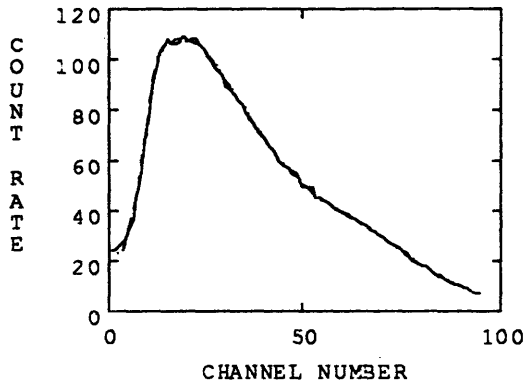

Figure 3.3: The first spectrum from scan sd133 (solid curve) and its reconstruction (dashed curve) from the first four principal components computed from ensemble sd133.

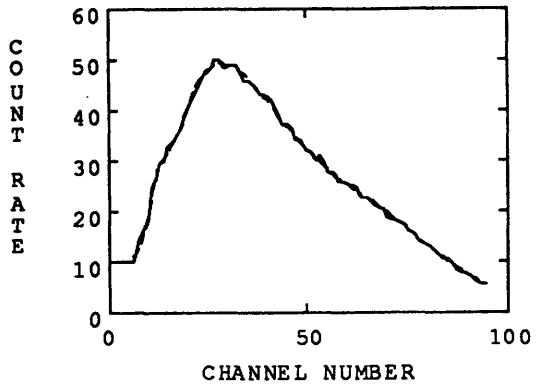

Figure 3.4: The 60th spectrum from scan sd133 (solid curve) and its reconstruction (dashed curve) from the first four principal components computed from ensemble sd133.

3.3 PHS Reconstructions with Principal Components

From the previous section, a PHS can be well reconstructed with only a few principal components. In fact, if all n independent principal components are used to reconstruct a n-dimensional PHS, the reconstruction should be perfect under all possible circumstances. However, the condition for having good reconstructions with only a few principal components is that the gain and offset for the PHS and the principal components must be the same. As long as this condition holds, we can get good PHS reconstructions from only four or six principal components no matter how the shapes of the PHS might change due to the physical property variation of the rock formation. Figure 3.5 and Figure 3.6 show two spectra from scan sd133a and their reconstructions with the first four principal components from scan sd133. Although the spectra in scan sd133a are not included in the computation of the principal components, good reconstructions are obtained. For all the spectra and principal components, the gain is 3.2 keV / channel and the offset is 0.0 keV.

Once the gain and offset of a PHS do not match with the gain and offset of the principal components, the PHS cannot be well reconstructed with only a few principal components. In Figure 3.7, the solid spectrum is the obtained by mapping the first spectrum in scan sd133 into a new gain (2.9 keV / channel) and offset (-3.2 keV). The reconstruction of the mapped spectrum with the first four principal components from sd133 is the dashed curve. Compared with Figure 3.3, the reconstruction in Figure 3.7 does not match the original well. Another example of bad reconstruction is shown in Figure 3.8 where the solid curve is obtained by mapping the sixtieth spectrum in sd133 into a 2.9 keV/channel gain and -3.2 keV offset. Compared with Figure 3.4, the reconstruction in Figure 3.8 does not match the original well.

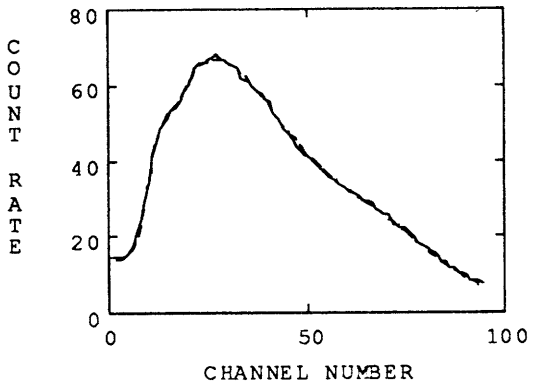

Figure 3.5: The 70th spectrum from scan sd133a (solid curve) and its reconstruction (dashed curve) from the first four principal components computed from ensemble sd133.

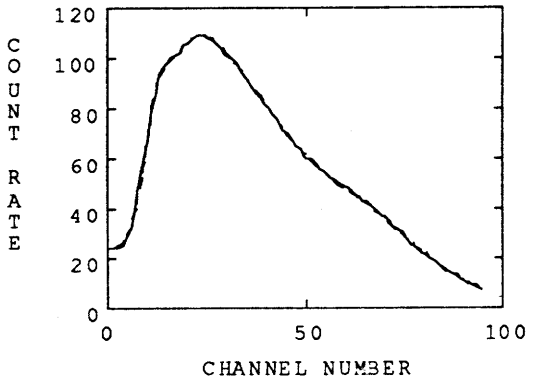

Figure 3.6: The 150th spectrum from scan sd133a (solid curve) and its reconstruction (dashed curve) from the first four principal components computed from ensemble sd133.

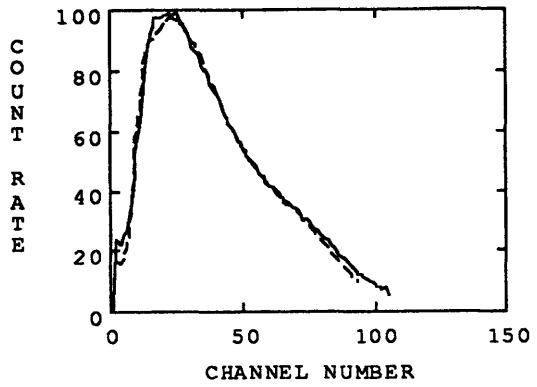

Figure 3.7 Solid spectrum: The first spectrum in sd133 mapped into gain = 2.9 keV/channel, offset = -3.2 keV; Dashed spectrum: reconstruction from the first four principal components computed from sd133.

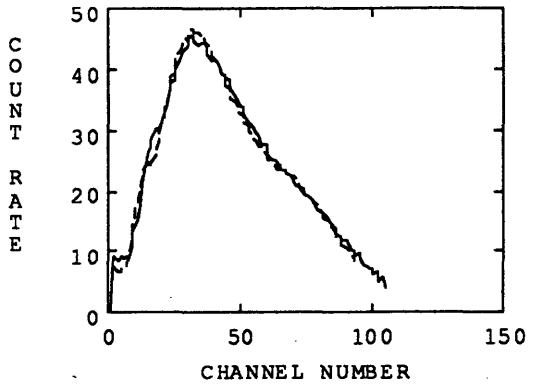

Figure 3.8 Solid spectrum: The sixtieth spectrum in sd133 mapped into gain = 2.9 keV/channel, offset = -3.2 keV; Dashed spectrum: reconstruction from the first four principal components computed from sd133.

Based on the PHS reconstruction behavior, we can determine whether or not the gain and offset of a PHS matches with the gain and offset of a set of principal components:

If a PHS can be well reconstructed with a few (four to six) principal components, then the PHS and the set of principal components have the same gain and offset.

If a PHS cannot be well reconstructed with a few principal components, then the PHS and the set of principal components have different gain and offset.

Further, we can estimate the gain and offset of a PHS based on the reconstruction behavior. Given a pair of gain and offset (g, o), a set of principal components for the PHS can be computed. Given a PHS with unknown gain and offset, reconstructions can be computed with different set of principal components with known gain and offset (g, o) until a good enough reconstruction is found. The gain and offset for the set of principal components which give the good reconstruction are then the estimated gain and offset for the PHS.

In implementing the estimation, instead of computing the principal components for different pairs of gain and offset (g, o), we keep the principal components with standard gain and offset (g_0, o_0) unchanged. Because all the PHS used in our analysis are acquired with gain 3.2 keV / channel and offset 0.0 keV, we use this gain and offset as our standard gain and offset. Given a PHS with unknown gain and offset, we mapped the PHS from a pair of guessed gain and offset (g, o) into the standard gain and offset, and compute the reconstruction of the mapped PHS with the principal components. If our guessed gain and offset (g, o) match the gain and offset for the PHS, we should get a good reconstruction.

Chapter 4 The Algorithm

The problem we are trying to solve in our research is: given a pulse height spectrum measured by a gamma ray device, determine the gain and offset for the PHS without using any stabilization radiation sources.

From the discussion in Chapter 3, a PHS can be well reconstructed with a few (four to six) principal components if and only if the gain and offset for the PHS and the set of principal components match. In this chapter, we develop an estimation algorithm for gain and offset based on the PHS reconstruction behavior. Our algorithm includes the following steps:

Step 1.	Define a set of four principal components with the standard gain and offset (g_0, o_0)[11].
Step 2.	For every pair of gain and offset (g, o) in a given range, map the given PHS from (g, o) into the standard gain and offset (g_0, o_0).
Step 3.	Reconstruct the mapped PHS with the set of principal components defined in Step 1.
Step 4.	Define cost functions which determine how good a reconstruction is. A cost function is minimized at the best reconstruction.
Step 5.	Search for the pair of gain and offset at which the principal components give the best reconstruction. The resulted pair of gain and offset is our estimation for the given PHS.

First, we discuss some of the preliminaries on PHS gain and offset changes. Then we introduce our estimation algorithm, the problems encountered in implementation and how we solve the problems.

---

[11] We define the standard gain and offset as: $g_0 = 3.2$ keV / channel; $o_0 = 0.0$ keV. All PHS in Table 1 are acquired with the standard gain and offset.

4.1 Some Preliminaries on Gain and Offset Changes

A pulse height spectra measured by a scintillator detector contains multichannel representing different energy range. Let f(E) be the number of photons with energy E detected per unit energy per second. Then f(E)dE is the number of photons with energy between E and E+dE detected within a second, the pulse height spectrum in energy space. A pulse height spectrum is registered in "channel space" by means of a coordinate transformation:

$$E = g\,x + o \qquad < 4.1\text{-}1 >$$

where x is the channel number, g is gain (keV/channel) and o is offset (keV). Take the derivative of equation < 4.1-1 >,

$$dE = g\,dx \qquad < 4.1\text{-}2 >$$

Therefore, we get the following:

$$f(E)\,dE = g\,f(g\,x + o)\,dx \equiv z(x)\,dx \qquad < 4.1\text{-}3 >$$

where

$$z(x) \equiv g\,f(g\,x + o) \qquad < 4.1\text{-}4 >$$

z(x) is the pulse height spectrum in channel space with gain and offset (g, o). All the spectra in our data ensemble are PHS in channel space.

Now suppose that the pulse height spectrum is acquired with a different pair of gain and offset (g', o'), the following is true:

$$f(E)\,dE = g\,f(g\,x + o)\,dx = g'\,f(g'\,x' + o')\,dx' \qquad < 4.1\text{-}5 >$$

and

$$E = g\,x + o = g'\,x' + o' \qquad < 4.1\text{-}6 >$$

must also be true. Let

$$h(x') = g' f(g' x' + o') \qquad < 4.1\text{-}7 >$$

h(x') is then the pulse height spectrum with gain and offset (g', o'). With equation <4.1-4> and <4.1-7>, we get the following from equation <4.1-5>:

$$z(x)\,dx = h(x')\,dx' \qquad < 4.1\text{-}8 >$$

From equation <4.1-6>, we get

$$x = \frac{g' x' + o' - o}{g} = \frac{g'}{g} x' + \frac{o' - o}{g} \qquad < 4.1\text{-}9 >$$

and

$$dx = \frac{g'}{g} dx' \qquad < 4.1\text{-}10 >$$

Substitute x and dx in equation < 4.1-8 > with equation < 4.1-9 > and < 4.1-10 >, we get the transformation from a PHS z(x) with the pair of gain and offset (g, o) to a equivalent PHS h(x') with another pair of gain and offset (g', o'):

$$h(x') = \frac{g'}{g}\, z\left(\frac{g'}{g} x' + \frac{o' - o}{g}\right) = \frac{g'}{g}\, z(x) \qquad < 4.1\text{-}11 >$$

This is the transformation used in the spectrum mapping program mentioned earlier.

The transformation in < 4.1-11 > is reversible, that is, if we transform h(x') from gain and offset (g', o') to (g, o), we get back the PHS z(x). Let z'(x) be the PHS with gain and offset (g, o) transformed from h(x') with (g', o'), then

$$z'(x) = \frac{g}{g'}\, h\left(\frac{g}{g'} x + \frac{o - o'}{g'}\right) \qquad < 4.1\text{-}12 >$$

Let $x = \frac{g' x' + o' - o}{g} = \frac{g'}{g} x' + \frac{o' - o}{g}$

equation < 4.1-12 > becomes

$$z'(x) = \frac{g}{g'} h\left(\frac{g}{g'}\left[\frac{g'}{g}x' + \frac{o' - o}{g}\right] + \frac{o - o'}{g'}\right) = \frac{g}{g'} h(x') = z(x) \qquad < 4.1\text{-}12 >$$

Therefore, we proved z'(x) = z(x). Note that z'(x) = z(x) is true only when z(x) is defined at all real values of x for $x \in [x_{min}, x_{max}]$.

4.2 Estimation Algorithms

Three estimation algorithms for estimating the gain and offset of a pulse height spectrum are discussed in this section. The three algorithms are: least square error (LSE) estimation, maximum likelihood (ML) estimation and maximum a posteriori (MAP) estimation. We used LSE and ML estimations in our study. Before discussing the estimation algorithms, we define the estimation problem in a mathematical manner.

Let h(x) be an observed pulse height spectrum (number of photons per channel per second) where x is the channel number. The gain and offset (g, o) for h(x) are unknown. Usually, PHS obtained from well logging contains noise most of which are due to the counting process. Let z(x, g, o) be the noise free part of h(x), then

$$h(x) = z(x, g, o) + noise(x) \qquad < 4.2\text{-}1 >$$

Let z(x) be the equivalent PHS of z(x, g, o) with standard gain and offset (g_0, o_0). From the transformation in equation < 4.1-11 >, we have

$$z(x, g, o) = \frac{g}{g_0} z\left(\frac{g}{g_0}x + \frac{o - o_0}{g_0}\right) \qquad < 4.2\text{-}2 >$$

As discussed in the previous chapter, z(x) can be fully constructed as:

$$z(x) = \sum_{i=1}^{m} a_i u_i(x) + \{z(x)\} \qquad < 4.2\text{-}3 >$$

where u_i's are the principal components with standard gain and offset (g_0, o_0), a_i's are the corresponding reconstruction coefficients, $\langle z(x) \rangle$ is the average PHS with (g_0, o_0), and m is between four to six. If we take the transformation in equation <4.1-11> on both sides of <4.2-3>, we get:

$$z(x, g, o) = \sum_{i=1}^{m} a_i u_i(x, g, o) + \langle z(x, g, o) \rangle \qquad < 4.2\text{-}4 >$$

where $u_i(x, g, o)$ is the transformed principal component,

$$u_i(x, g, o) = \frac{g}{g_0} u_i\left(\frac{g}{g_0} x + \frac{o - o_0}{g_0}\right) \qquad < 4.2\text{-}5 >$$

and $\langle z(x, g, o) \rangle$ is the average PHS with gain and offset (g, o),

$$\langle z(x, g, o) \rangle = \frac{g}{g_0} \left\langle z\left(\frac{g}{g_0} x + \frac{o - o_0}{g_0}\right) \right\rangle \qquad < 4.2\text{-}6 >$$

Equation < 4.2-4 > can be written as

$$z(x, g, o) = U(x, g, o)\, a + \langle z(x, g, o) \rangle \qquad < 4.2\text{-}7 >$$

where $U(x, g, o)$ is a matrix that contains m principal components,

$$U(x, g, o) = [u_1(x, g, o)\ u_2(x, g, o) \cdots u_m(x, g, o)] \qquad < 4.2\text{-}8 >$$

and a is a vector that contains the reconstruction coefficients,

$$a = [a_1\ a_2 \cdots a_m]^T \qquad < 4.2\text{-}9 >$$

The PHS h(x) can then be written as:

$$h(x) = U(x, g, o)\, a + \{ z(x, g, o) \} + \text{noise}(x) \qquad < 4.2\text{-}10 >$$

Now the estimation problem becomes: given the observed pulse height spectrum h(x) , determine the gain and offset (g, o) and the reconstruction coefficient vector **a**.

In the three estimation algorithms, we show how the reconstruction coefficient vector **a** is computed, and how the cost functions are defined as the criterion for gain and offset estimation.

4.2a Least Square Error Estimation

For a given pulse height spectrum h(x) with unknown gain and offset (g', o'), the estimated PHS with m principal components is:

$$\hat{z}(x, g, o) = U(x, g, o)\, a + \{z(x, g, o)\} \qquad < 4.2a\text{-}1 >$$

Without taking the noise part of h(x) into account, the reconstruction coefficient vector **a** is computed so that $\delta^2(\hat{z})$,the square error between h(x) and $\hat{z}(x, g, o)$ defined below is minimized.

$$\delta^2(\hat{z}) = (\hat{z} - h)^T (\hat{z} - h) \qquad < 4.2a\text{-}2 >$$

From Chapter 3, **a** is then the projection coefficients for projecting h(x) onto the principal components.

$$a = U(x, g, o)^T (h(x) - \{z(x, g, o)\}) \qquad < 4.2a\text{-}3 >$$

If (g, o), the gain and offset for the principal components and the mean spectrum matches with (g', o'), then the estimated PHS $\hat{z}(x, g, o)$ should match with the given PHS h(x) well. The mean square error $\delta^2(\hat{z})$ is a good criterion for judging how well the estimated PHS $\hat{z}(x, g, o)$ fits the original PHS h(x). However, $\delta^2(\hat{z})$ weights every channel in the PHS equally while the counts in different channels, and therefore, the noise level could vary significantly. In the weighted least square error estimation, we modified equation <4.2a-2> and define the cost function as the weighted mean square error:

$$\varepsilon^2(g, o) = \frac{1}{N}\sum_{x=1}^{N} \frac{(\hat{z}(x, g, o) - h(x))^2}{\hat{z}(x, g, o)} \qquad < 4.2a\text{-}4 >$$

where $\frac{1}{\hat{z}(x, g, o)}$ is the weighting function. We will see in section 4.2b that $\hat{z}(x, g, o)$ is approximately the noise variance of h(x) in each channel x. The reconstruction coefficient vector defined by equation <4.2a-3> does not necessarily minimize the above cost function $\varepsilon^2(g, o)$. Further investigations need to be done on the computation of **a**. In Section 4.2b, we introduce another way of computing **a**.

The pair of gain and offset (g, o) that minimizes $\varepsilon^2(g, o)$ is our estimated gain and offset for (g', o'). For computing the cost function < 4.2a-4 > at each pair of gain and offset (g, o), the transformed principal component matrix $U(x, g, o)$ and the transformed average PHS $\{z(x, g, o)\}$ are required. However, instead of computing **a** and ε^2 by transforming the principal components and average spectrum from the standard gain and offset (g_0, o_0) to (g, o), it is numerically simpler to leave U(x) and $\{z(x)\}$ at (g_0, o_0) and transform h(x) from (g, o) to (g_0, o_0). Because of the discrete nature of the channels in the spectra, and the need to interpolate when performing the transformation (see Section 4.3), the **a** and ε^2 computed at (g, o) may not be numerically identical to the ones computed at (g_0, o_0), but we believe that these effects will not be very important for our estimation algorithms, Thus, let $h_0(x)$ be the PHS transformed from h(x) with (g, o) into $h_0(x)$ with the standard gain and offset (g_0, o_0)

$$h_0(x) = \frac{g_0}{g}\, h\left(\frac{g_0}{g}x + \frac{o_0 - o}{g}\right) \qquad < 4.2a\text{-}5 >$$

Now transform both sides of equation <4.2a-3 > from gain and offset (g, o) into the standard gain and offset (g_0, o_0). On the left side, **a** should remain approximately the same after the transformation. On the right side, $U(x, g, o)$ becomes U(x) which is the matrix contains the principal components u_i's with the standard gain and offset, h(x) becomes $h_0(x)$, and $\{z(x, g, o)\}$ becomes $\{z(x)\}$. Therefore, equation < 4.2a-3 > becomes the following after the transformation:

$$a = U(x)^T (h_0(x) - \{z(x)\}) \qquad < 4.2a\text{-}6 >$$

Let $\hat{z}(x)$ be the transformation of $\hat{z}(x, g, o)$

$$\hat{z}(x) = \frac{g_0}{g}\,\hat{z}\left(\frac{g_0}{g}x + \frac{o_0 - o}{g}, g, o\right)$$

Take the transformation on both sides of equation < 4.2a-1 >, we get

$$\hat{z}(x) = U(x)\,a + \langle z(x) \rangle \qquad < 4.2a\text{-}7 >$$

We can then keep the principal component matrix $U(x, g, o)$ and average spectrum $\langle z(x, g, o) \rangle$ with (g_0, o_0) unchanged and compute the cost function with the transformed PHS:

$$\varepsilon_L^2(g, o) = \frac{1}{N}\sum_{x=1}^{N} \frac{(\hat{z}(x) - h_0(x))^2}{\hat{z}(x)} \qquad < 4.2a\text{-}8 >$$

In the implementation of the least square error estimation as well as the estimation algorithms in the next sections, only the given PHS is transformed.

For a given PHS h(x), the estimation for gain and offset with the least square error estimation algorithm is done in the following steps:

1. Take a guess for the gain and offset, (g, o).

2. Map h(x) from (g, o) into (g_0, o_0) to get the spectrum $h_0(x)$:

$$h_0(x) = \frac{g_0}{g}\,h\left(\frac{g_0}{g}x + \frac{o_0 - o}{g}\right)$$

3. Reconstruct $h_0(x)$ with the first m (m is four) principal components to get $\hat{z}(x)$:

$$\hat{z}(x) = U(x)\,a + \langle z(x) \rangle$$

where **a** is the reconstruction coefficient vector:

$$a = U(x)^T\,(h_0(x) - \langle z(x) \rangle)$$

4. Compute the cost function, that is the weighted mean square error:

$$\varepsilon^2(g, o) = \frac{1}{N}\sum_{x=1}^{N}\frac{(\hat{z}(x) - h_0(x))^2}{\hat{z}(x)}$$

where N is the number of channels in z(x) and $\hat{z}(x)$.

5. Find (g, o) that minimizes the above cost function. This step will be discussed later in this chapter.

Figure 4.1 shows an example for implementing the above five steps. Figure 4.2 shows a plot of the cost function vs gain which has a minimum around the true gain.

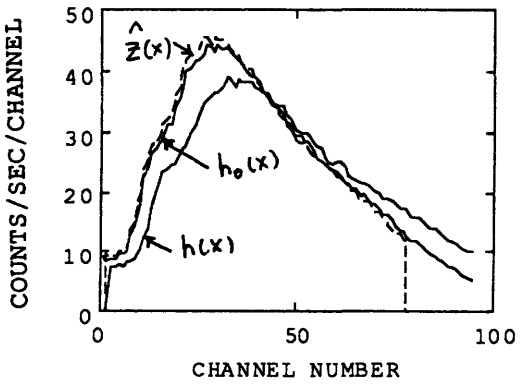

Figure 4.1: Estimation for gain and offset
h(x): the given PHS with unknown gain and offset.
$h_0(x)$: the transformation of h(x) from (g, o) into (g_0, o_0).
$\hat{z}(x)$: the principal components estimation of $h_0(x)$.

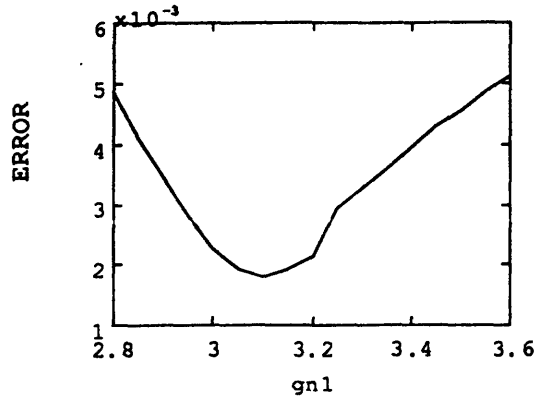

Figure 4.2: The cost function (< 4.2a-8 >) vs. gain for a spectrum in ensemble sd133.

4.2b Maximum A Posteriori (MAP) and Maximum Likelihood (ML) Estimation

The weighted least squares algorithm proposed in the proceeding section is closely related to maximum likelihood (ML) estimation of the gain and offset. ML estimation, in turn, can be generalized to maximum a posteriori (MAP) estimation if prior information on the probability distribution of the gain, offset and the coefficient vector **a** for principal component reconstruction are available.

Let h(x) be the observed PHS containing Poisson noise, and x is channel number. The gain and offset (g, o) for h(x) is unknown. The MAP estimation problem becomes: given h(x), determine g, o and **a** such that the conditional probability $p(g, o, \mathbf{a} \mid h(x))$ is maximized.

From Bayes' theorem:

$$p(g, o, \mathbf{a} \mid h(x)) = \frac{p(h \mid g, o, \mathbf{a})\, p(g, o, \mathbf{a})}{p(h)} \qquad < 4.2b\text{-}1 >$$

Taking the logarithm of both sides, the problem is equivalent to maximizing:

$$\ln[\, p(\, g, o, a \mid h(x))\,] = \ln[\, p(h \mid g, o, a)\,] + \ln[\, p(g, o, a)\,] - \ln[\, p(h)\,] \qquad < 4.2b\text{-}2 >$$

The last term on the right side does not depend on g, o or a, therefore it can be dropped. Further, we can assume that g, o and a are independent, and the joint probability for g, o, and a is the product of the unconditional individual probabilities:

$$p(g, o, a) = p(g)\, p(o)\, p(a). \qquad <4.2b\text{-}3 >$$

and

$$\ln[\, P(\, g, o, a)] = \ln[\, P(g)\,] + \ln[\, P(o)\,] + \ln[\, P(a)\,]. \qquad <4.2b\text{-}4 >$$

The MAP estimation problem becomes: finding g, o and a which maximize

$$-\varepsilon^2(g, o, a) = \ln[\, P(h \mid g, o, a)] + \ln[\, P(g)\,] + \ln[\, P(o)\,] + \ln[\, P(a)\,] \qquad < 4.2b\text{-}5 >$$

which is equivalent to find g, o, and a which minimize the following:

$$\varepsilon^2_{MAP}(g, o, a) = -\,\{\, \ln[\, P(h \mid g, o, a)] + \ln[\, P(g)\,] + \ln[\, P(o)\,] + \ln[\, P(a)\,]\,\} \qquad < 4.2b\text{-}6 >$$

Equation < 4.2b-6 > is then the cost function for maximum a posteriori (MAP) estimation.

The first term on the right side of equation < 4.2b-5 >, $\ln[\, P(h \mid g, o, a)]$, is the logarithm of the likelihood function and the remaining three terms are prior distributions. If we maximize only the likelihood function, we are making a maximum likelihood (ML) estimation. Therefore, the cost function for maximum likelihood estimation is:

$$\varepsilon^2_{ML}(g, o, a) = -\,\ln[\, P(h \mid g, o, a)] \qquad < 4.2b\text{-}7 >$$

Now the cost functions for MAP and ML estimation are defined. Next, we consider different terms in the cost functions separately.

For the likelihood function, let's examine the spectrum h(x). Usually, PHS obtained from the well logging field contain noise. Recall equation < 4.2-1 >,

$$h(x) = z(x, g, o) + noise(x)$$

where $z(x, g, o)$ is the noise free part of h(x) and $noise(x)$ is the noise part which is zero mean and uncorrelated between channels. Most of the noise comes from the counting process, and therefore we can use the Poisson distribution to model the noise. If we average h(x) over the noise, we get:

$$\langle h(x) \rangle = z(x, g, o). \qquad < 4.2b\text{-}8 >$$

Therefore, h(x) is distributed as a Poisson random variable with mean $z(x, g, o)$:

$$p(h \mid g, o, a) = \prod_{x=1}^{n} \frac{z^h e^{-z}}{h!} \qquad < 4.2b\text{-}9 >$$

and

$$\ln[\, p(h \mid g, o, a)\,] = \sum_{x=1}^{N} [\, h(x)\, \ln[\, z(x, g, o)\,] - z(x, g, o) - \ln[\, h(x)\,!\,]\,] \qquad < 4.2b\text{-}10 >$$

Recall equation < 4.2-7 >,

$$z(x, g, o) = U(x, g, o)\, a + \langle z(x, g, o) \rangle \qquad < 4.2b\text{-}11 >$$

the reconstruction coefficient vector **a** dependence of $z(x, g, o)$ separates from the gain and offset (g, o) dependence, and $z(x, g, o)$ is linear in **a**. For any pair of gain and offset (g, o), let us determine the optimal reconstruction coefficient vector **a** that maximizes the likelihood function $\ln[\, p(h \mid g, o, a)\,]$

$$\frac{\partial \ln[\, p(h \mid g, o, a)\,]}{\partial a_i} = \sum_{x=1}^{N} \left[\frac{h(x)}{z(x, g, o)} \frac{\partial\, z(x, g, o)}{\partial a_i} - \frac{\partial\, z(x, g, o)}{\partial a_i} \right] = 0 \qquad < 4.2b\text{-}12 >$$

$1 \le i \le m$, a_i is a component of **a**.

From equation < 4.2b-11 >,

$$\frac{\partial\ z(x, g, o)}{\partial\ a_i} = u_i(x, g, o)$$

Equation < 4.2b-12 > becomes:

$$\sum_{x=1}^{N} \left[\frac{h(x)\, u_i(x, g, o) - u_i(x, g, o)\ z(x, g, o)}{z(x, g, o)} \right] = 0$$

We can approximate the denominator $z(x, g, o)$ with h(x), and use the definition in equation < 4.2b-11 > to get:

$$\sum_{x=1}^{N} \left[\frac{[\, h(x) - \{ z(x, g, o) \}]\, u_i(x, g, o) - u_i(x, g, o)\ U(x, g, o)\, a_i}{h(x)} \right] = 0 \qquad < 4.2b\text{-}13 >$$

Let H be a diagonal matrix whose diagonal elements are h(x):

$$H = \begin{bmatrix} h(1) & 0 & \\ 0 & h(2) & \\ & & \cdots \end{bmatrix}$$

Equation < 4.2b-13 > can now be written as a matrix equation:

$$U^T(x, g, o)\ H^{-1}[\, h(x) - \{ z(x, g, o) \}] - U^T(x, g, o)\ H^{-1}\, U(x, g, o)\, a = 0$$

which gives:

$$a(g, o) = [\, U^T(x, g, o)\ H^{-1}\, U(x, g, o)]^{-1}\, U^T(x, g, o)\ H^{-1}[\, h(x) - \{ z(x, g, o) \}] \qquad < 4.2b\text{-}14 >$$

a(g, o) is the optimal estimate of a for any given gain and offset (g, o). Using a(g, o), we can estimate the noise free PHS z(x, g, o) as:

$$\hat{z}(x, g, o) = U(x, g, o)\, a(g, o) + \{ z(x, g, o) \} \qquad < 4.2b\text{-}15 >$$

The likelihood function becomes a function with only gain and offset (g, o) as variables:

$$\ln[\,p(h \mid g, o, a)\,] = \sum_{x=1}^{N} [\,h(x)\,\ln[\,\hat{z}(x, g, o)\,] - \hat{z}(x, g, o) - \ln[\,h(x)!\,]\,] \qquad < 4.2b\text{-}16 >$$

In implementing the estimation algorithm, we compute the likelihood function differently than equation < 4.2b-16 >. As discussed in the least square estimation, instead of computing the cost functions with the principal components and average spectrum transformed into with different gain and offset (g, o), we compute the cost function with the principal components U(x) and average spectrum $\{z(x)\}$ at the standard gain and offset. And we use $h_0(x)$, the transformation of h(x) from (g, o) into the standard gain and offset:

$$h_0(x) = \frac{g_0}{g}\, h\left(\frac{g_0}{g}x + \frac{o_0 - o}{g}\right) \qquad < 4.2b\text{-}17 >$$

As a result, the above equations are rewritten correspondingly as

$$h_0(x) = z(x) + noise_0(x)$$

$$z(x) = U(x)\,a + \{z(x)\}$$

$$H = \begin{bmatrix} h_0(1) & 0 & \\ 0 & h_0(2) & \\ & & \cdots \end{bmatrix}$$

Then the optimal a is

$$a(g, o) = \left[\,U^T(x)\,H^{-1}\,U(x)\,\right]^{-1} U^T(x)\,H^{-1}\left[\,h_0(x) - \{z(x)\}\,\right] \qquad < 4.2b\text{-}18 >$$

and the estimated noise free PHS with the standard gain and offset is

$$\hat{z}(x) = U(x)\,a(g, o) + \{z(x)\} \qquad < 4.2b\text{-}19 >$$

We expect that there may be numerical effects associated with applying the Poisson model to $h_0(x)$ instead of h(x), but these have not been investigated.

Let H' be the identity matrix I scaled with a constant:

$$H' = \text{constant} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ & \cdots \end{bmatrix}$$

If we replace H with H', equation <4.2b-18> becomes the same as <4.2a-6>. We call the **a** computed from <4.2a-6> the projection coefficient vector, and **a**(g, o) computed from <4.2b-18> the optimal coefficient vector. We expect **a**(g, o) to give better estimation results than **a**. However, in Chapter 5 the estimation performance tests show that **a**(g, o) is not any better than **a**. The reason is unknown.

With $\hat{z}(x)$ defined in equation <4.2b-19>, the likelihood function is approximately (see the discussion following equation <4.2a-4>):

$$\ln[\, p(h_o \mid g, o, a)\,] = \sum_{x=1}^{N} [\, h_o(x)\ \ln[\, \hat{z}(x)\,] - \hat{z}(x) - \ln[\, h_o(x)\,!\,]\,] \qquad < 4.2b\text{-}20 >$$

The cost function (< 4.2b-7 >) for maximum likelihood estimation becomes

$$\varepsilon^2_{ML}(g, o) = -\ln[\, p(h_o \mid g, o, a)\,] = -\sum_{x=1}^{N} [\, h_o(x)\ \ln[\, \hat{z}(x)\,] - \hat{z}(x) - \ln[\, h_o(x)\,!\,]\,] \qquad < 4.2b\text{-}21 >$$

For maximum likelihood estimation, in some of our tests the projection coefficients (< 4.2a-6 >) will be used to replace a(g, o):

$$a = U(x)^T (h_o(x) - \{z(x)\}) \qquad < 4.2b\text{-}22 >$$

The cost function (< 4.2b-6 >) for maximum a posteriori estimation of gain and offset can be formed with the likelihood function (< 4.2b-21 >) and the apriori information

$$\varepsilon^2_{MAP}(g, o) = -\sum_{x=1}^{N} [\, h_o(x)\ \ln[\, \hat{z}(x)\,] - \hat{z}(x) - \ln[\, h_o(x)\,!\,]\,] + \ln[\, P(g)\,] + \ln[\, P(o)\,] \qquad < 4.2b\text{-}23 >$$

In our study, we only implemented the maximum likelihood estimation.

For a given PHS h(x), the estimation for gain and offset with ML estimation algorithm is done in the following steps:

1. Take a guess for the gain and offset, (g, o).

2. Compute a or $a(g, o)$ and $\hat{z}(x)$

3. Compute the cost function for ML estimation[12].

4. Find (g, o) that minimizes the cost function. This step will be discussed later in this chapter.

4.2c Connections Between the Estimation Algorithms

The procedures for the three estimation algorithms discussed above are the same. The difference among the estimation algorithms is the definition of the cost functions and the computation of reconstruction coefficient vector **a**. The reconstruction coefficient vector can be either the projection coefficient vector as in equation < 4.2b-22 >, or the optimal coefficient vector as in equation < 4.2b-18 > that maximizes the likelihood function.

Once **a** is defined, we can get the reconstructed PHS with the principal components and then computed the cost functions. The cost function for the weighted least square estimation is defined in equation < 4.2a-8 >:

$$\varepsilon_L^2(g, o) = \frac{1}{N} \sum_{x=1}^{N} \frac{(\hat{z}(x) - h_0(x))^2}{\hat{z}(x)} \qquad < 4.2c\text{-}1 >$$

The cost function for maximum likelihood estimation is defined in equation <4.2b-25>:

$$\varepsilon_{ML}^2(g, o) = -\ln[\, p(h_0 \mid g, o, \hat{a})\,] = -\sum_{x=1}^{N} [\, h_0(x) \ln[\,\hat{z}(x)\,] - \hat{z}(x) - \ln[\, h_0(x)!\,]\,]$$

---

12 The cost function for maximum likelihood estimation is defined in equation <4.2b-21>. In implementing the algorithm, we used the simplified cost function in equation <4.2c-2> in Section 4.2c.

When $h_0(x) >> 1$ and $|\hat{z}(x) - h_0(x)| << h_0(x)$, as is generally the case for our PHS, the Poisson distribution approaches a normal distribution and the likelihood function can be approximated as:

$$\sum_{x=1}^{N} \left[h_0(x) \ln[\hat{z}(x)] - \hat{z}(x) - \ln[h_0(x)!] \right] \cong \sum_{x=1}^{N} \left[-\frac{|\hat{z}(x) - h_0(x)|^2}{2\hat{z}(x)} - \frac{1}{2} \ln[2\pi \hat{z}(x)] \right]$$

The cost function for maximum likelihood estimation can the be written as:

$$\varepsilon^2_{ML}(g, o) \cong \frac{N}{2} \varepsilon^2(g, o) + \frac{1}{2} \sum_{x=1}^{N} \ln[2\pi \hat{z}(x)] \qquad < 4.2c\text{-}2 >$$

and the cost function for maximum a posteriori estimation (equation < 4.2b-23 >) can be written as:

$$\varepsilon^2_{MAP}(g, o) = \varepsilon^2_{ML}(g, o) + \ln[P(g)] + \ln[P(o)] \qquad < 4.2c\text{-}3 >$$

4.3 The Cost Function Behavior

A cost function should be smooth and have a clear global minimum around the true gain and offset. However, in the implementation, the cost functions show some strange behaviors as shown in Figure 4.3 and 4.4. In Figure 4.3, the cost function has glitches, but there is a global minimum. In Figure 4.4, it is not clear if the cost function has a global minimum or not. The cost function decreases quickly as gain reaches small values. These problems are possibly caused by the mapping subroutine that implements the PHS transformation from one pair of gain and offset to another, or by truncation effects. These effects will be discussed in this section.

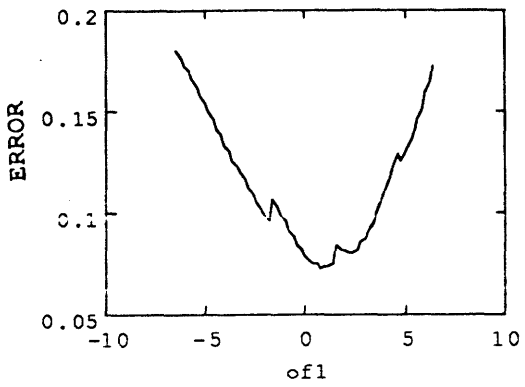

Figure 4.3 Cost function vs offset.
The first spectrum in sd133 is mapped into gain = 3.2 keV/channel, offset = 1.6 keV. The cost function for least square error estimation for offset is computed at different offset. The minimum occurs at of1 = 1.2 keV

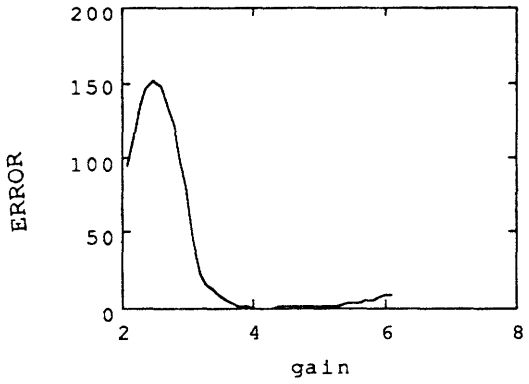

Figure 4.4 Cost function vs. gain
The fiftieth spectrum in sd133 is mapped into gain = 4.0 keV/channel, offset = 0.0 keV. The cost function for least square error estimation for gain is computed at different gain. The minimum occurs at gain = 4.1 keV/channel.

4.3a The Mapping Subroutines

In implementing the estimation algorithms, the mapping subroutines transform a PHS from one pair of gain and offset to another with the transformation in equation <4.1-11>:

$$h(x') = \frac{g'}{g}\, z\left(\frac{g'}{g}\, x' + \frac{o' - o}{g}\right) = \frac{g'}{g}\, z(x) \qquad \text{<4.3a-1>}$$

Where z(x) is the spectrum with gain and offset (g, o) being transformed to h(x') with gain and offset (g', o'). All PHS contain only the channels where the channel numbers are integers. In the above transformation, for some or all integer x', the corresponding x:

$$x = \frac{g'}{g}\, x' + \frac{o' - o}{g}$$

is a non-integer number. In order to compute h(x') from z(x), we must interpolate z(x) when x is not an integer.

There is a Fortran subroutine[13] that uses fourth order polynomial interpolation for z(x):

$$z(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \qquad \text{< 4.3a-2 >}$$

For each channel x' in h(x'), the corresponding channel in z(x) is x. Let xi be the integer part of (x + 0.5), the five points in z(x): z(xi-2), z(xi-1), z(xi), z(xi+1) and z(xi+2) are used to determine the coefficients a_i's in the above fourth order polynomial. With the fourth order polynomial interpolation, z(x) is not a continuous function of gain and offset (g', o'). For a fixed x', the change in g' or o' causes x to change. As long as integer xi remains the same, the same five points in z are used to determine the polynomial coefficients, and z(x) is computed from the same polynomial. However, when the change in g' or o' is just enough to cause xi to jump to another integer, the polynomial coefficients are determined

---

[13] The Fortran subroutine was written by W. Frawley, J. Grau and A. Becker at Schlumberger-Doll Research, Ridgefield, CT.

by five different points in z. This discontinuity in the transformation causes the glitches in the cost function shown in Figure 4.3.

We can also use linear interpolation as shown in Figure 4.5. Let xi be the integer part of x, then x is in between xi and xi+1 and z(x) is interpolated as the point on the line that connects z(xi) and z(xi+1):

$$z(x) = z(xi) + (x - xi)[\, z(xi + 1) - z(xi)\,] \qquad \text{<4.3a-3>}$$

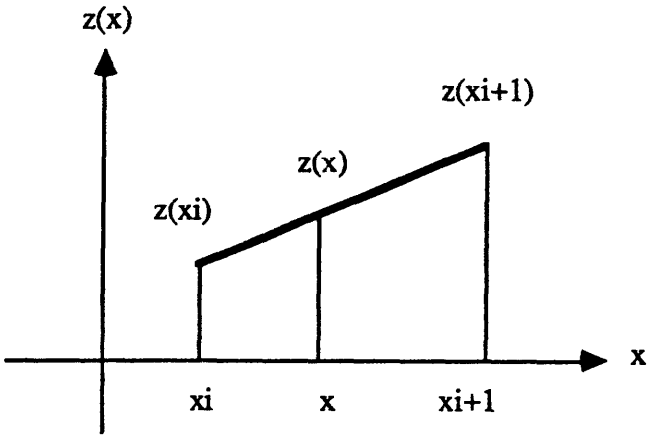

Figure 4.5 Two point linear interpolation

Figure 4.6 shows the cost function for least square error estimation computed with the linear interpolation.

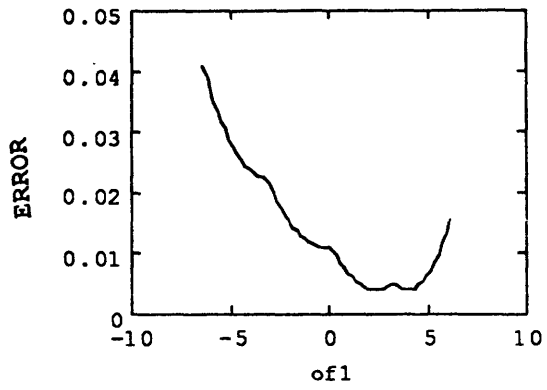

Figure 4.6 Cost function vs of1.
The first spectrum in sd133 is mapped into gain = 3.2 keV/channel, offset = 3.2 keV with linear interpolation. The cost function for least square error estimation for offset is computed at different offset. The minimum occurs at of1 = 3.5 keV

We want to have the interpolation vary smoothly with changes in gain and offset. From Shannon's sampling theorem, a band-limited function is completely determined by its discrete sampling. Assume the PHS z(x) is band limited, that is, the Fourier transform for z(x) in frequency domain $\tilde{Z}(w)$ is non-zero only for $|w|<B$ as shown in Figure 4.7.

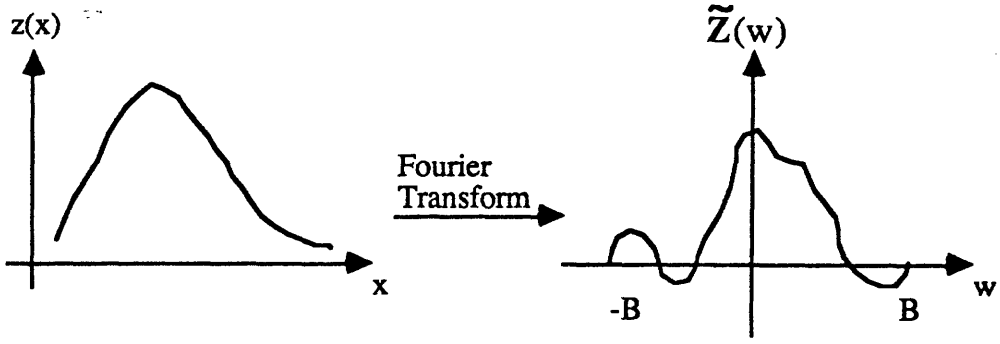

Figure 4.7 Assume z(x) is band-limited.

The Fourier transformation is:

$$\tilde{Z}(w) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} z(x)\, e^{iwx}\, dx \qquad \text{<4.3a-4>}$$

and

$$z(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \tilde{Z}(w)\, e^{-iwx}\, dw \qquad \text{<4.3a-5>}$$

For $|w| < B$, $\tilde{Z}(w)$ can be written as:

$$\tilde{Z}(x) = \sum_{n=-\infty}^{\infty} a_n\, e^{in\pi w/B} \qquad \text{<4.3a-6>}$$

where the coefficients are

$$a_n = \frac{1}{2B} \int_{-B}^{B} \tilde{Z}(w)\, e^{-in\pi w/B}\, dw \qquad \text{<4.3a-7>}$$

Combine equation <4.3a-7> and <4.3a-5>, we have

$$z(\frac{n\pi}{B}) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \tilde{Z}(w)\, e^{-in\pi w/B}\, dw = \frac{1}{\sqrt{2\pi}} \int_{-B}^{B} \tilde{Z}(w)\, e^{-in\pi w/B}\, dw = \frac{2B}{\sqrt{2\pi}}\, a_n$$

and

$$a_n = \frac{\sqrt{2\pi}}{2B}\, z(\frac{n\pi}{B})$$

equation <4.3a-6> becomes

$$\tilde{Z}(w) = \sum_{n=-\infty}^{\infty} \frac{\sqrt{2\pi}}{2B}\, z(\frac{n\pi}{B})\, e^{in\pi w/B} \qquad \text{<4.3a-8>}$$

and equation <4.3a-5> becomes

$$z(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left[\sum_{n=-\infty}^{\infty} \frac{\sqrt{2\pi}}{2B}\, z(\frac{n\pi}{B})\, e^{in\pi w/B} \right] e^{-iwx}\, dw = \frac{1}{2B} \sum_{n=-\infty}^{\infty} \left[z(\frac{n\pi}{B}) \int_{-B}^{B} e^{i\,(n\pi w/B\, -\, wx)}\, dw \right]$$

The integral part inside the summation is simply

$$\int_{-B}^{B} e^{j(n\pi w/B - wx)}\, dw = \frac{2\sin(n\pi - Bx)}{(\frac{n\pi}{B} - x)}$$

Now let $\Delta = \frac{\pi}{B}$ be the sampling interval (Nyquist interval), then $\frac{B}{2\pi}$ is the Nyquist frequency $\frac{B}{2\pi} = \frac{1}{2\Delta}$. If z(x) is band-limited, then it can be exactly expressed as

$$z(x) = \sum_{n=-\infty}^{\infty} z(n\Delta) \frac{\sin\left[\pi(n - \frac{x}{\Delta})\right]}{\pi(n - \frac{x}{\Delta})}$$

Since we only have the samples for z(x) at $x = n\Delta = 1, 2, \cdots, N$, ($\Delta = 1$), we might as well assume that z(x) is band-limited with $B = \pi$ and use the above formula. Let n be the integer channel number for z(x), then for any real number x between 1 and N, we have the interpolation

$$z(x) = \sum_{n=1}^{N} z(n) \frac{\sin\left[\pi(n - x)\right]}{\pi(n - x)}$$

which should vary smoothly with changes in gain and offset. The transformation function in <4.3a-1> can be written as

$$h(x') = \frac{g'}{g} \sum_{n=1}^{N} z(n) \frac{\sin\left[\pi\left(n - \frac{g'x' + o' - o}{g}\right)\right]}{\pi\left(n - \frac{g'x' + o' - o}{g}\right)} \qquad \text{<4.3a-9>}$$

This is the transformation used in the implementation of our estimation algorithms. In the next three figures, we can see that the cost function for the least square estimation is smooth.

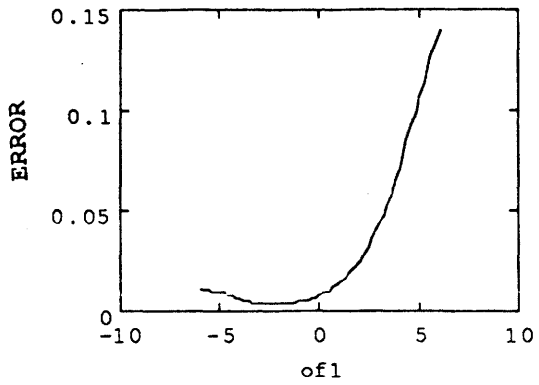

Figure 4.8 Cost function vs. offset
The first spectrum in sd133 mapped into gain = 3.2 keV/channel, offset = -2.0 keV. The cost function has minimum at of1 = -2.4 keV

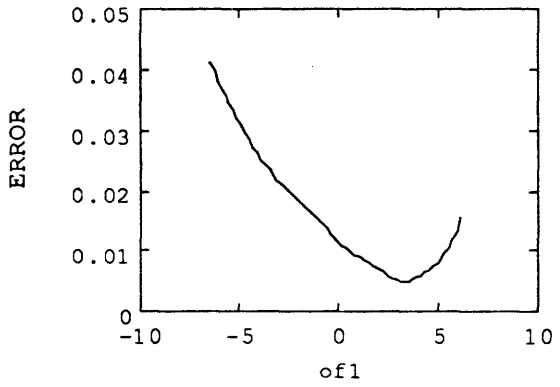

Figure 4.9 Cost function vs. offset
The first spectrum in sd133 mapped into gain = 3.2 keV/channel, offset = 3.2 keV. The cost function has minimum at of1 = 3.3 keV

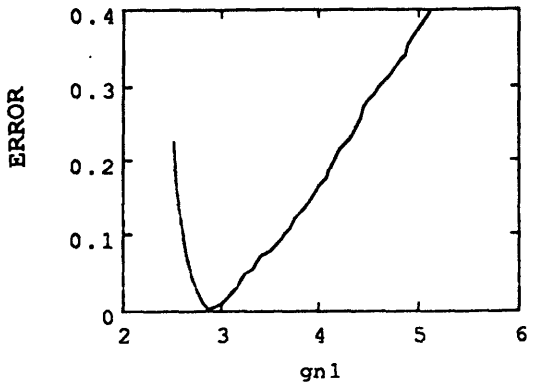

Figure 4.10 Cost function vs. gain
The first spectrum in sd133 mapped into gain = 2.9 keV/channel, offset = 0.0 keV. The cost function has minimum at of1 = 2.9 keV

All the three mapping subroutines using different interpolations are attached in Appendix 1.

4.3b The Truncation Effect

Suppose that the principal components are computed over the channels $n_{min} \le x \le n_{max}$. Let h(x) be a PHS with unknown gain and offset, and $h_0(x)$ is the PHS mapped from h(x) with a pair of gain and offset (g, o) into the standard gain and offset. Due to gain and offset (g, o) variations, $h_0(x)$ might be truncated so that $h_0(x)$ is non-zero only for the channels $n'_{min} \le x \le n'_{max}$ where

$$n_{min} \le n'_{min} \le x \le n'_{max} \le n_{max} \qquad < 4.3b\text{-}1 >$$

Then if we use the original set of principal components to compute $\hat{z}(x)$, the reconstruction of $h_0(x)$, the cost function will behave abnormally as shown in Figure 4.4 and lead to bad estimations. The PHS $h_0(x)$ should not be reconstructed from the original principal components because the original set of principal components does not form an orthogonal basis over the channels $n'_{min} \le x \le n'_{max}$ and therefore, does not give good reconstructions for $h_0(x)$. The PHS $h_0(x)$ should be reconstructed with the set of principal components that is computed over the channels $n'_{min} \le x \le n'_{max}$.

In order to avoid recomputing principal components for every n'_{min} and n'_{max}, we limit the variations in gain and offset (g, o) so that

$$n'_{min} \le m_{min} \le x \le m_{max} \le n'_{max} \qquad < 4.3b\text{-}2 >$$

The principal components are computed over the channels $m_{min} \le x \le m_{max}$, and $\hat{z}(x)$ is the principal component reconstruction of $h_0(x)$ over the channels $m_{min} \le x \le m_{max}$. The cost functions are then computed from $\hat{z}(x)$ and $h_0(x)$ over the channels $m_{min} \le x \le m_{max}$.

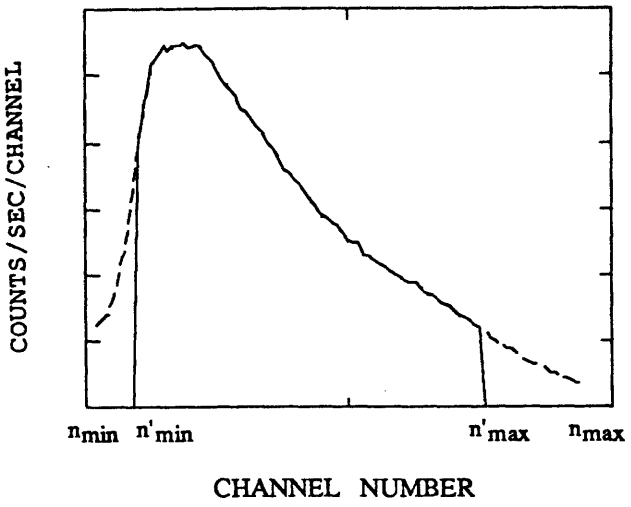

Figure 4.11 The first spectrum in sd133

In our analysis, the data we used are all acquired with the standard gain and offset. Let c(x) be a PHS from the scans listed in Table 1, and transform c(x) from the standard gain and offset (g_0, o_0) to a new gain and offset (g_h, o_h)

$$h(x) = \frac{g_h}{g_0} c\left(\frac{g_h}{g_0} x + \frac{o_h - o_0}{g_0}\right)$$

h(x) is then used as the PHS with unknown gain and offset. Implementing the estimation algorithms, we get the estimated gain and offset (g, o) for h(x). The performance of our estimation algorithms are tested by comparing the estimation (g, o) with the true gain and offset (g_h, o_h) for h(x). The variations in g_h and o_h have to be limited in order to satisfy the

condition described by equation <4.3b-2>. On one hand, we want g_h and o_h to vary in larger ranges to test how robust our algorithms are. On the other hand, we want the number of channels in the principal components $m_{max} - m_{min}$ to be as big as possible to contain enough information about the PHS. There is a trade off between $m_{max} - m_{min}$ and the variation range for g_h and o_h.

All the PHS used in our analysis contain 94 channels in each spectrum. The principal components are computed over 70 channels (channel 5 to channel 74) from the PHS. We allow the gain g_h to vary from 2.8 keV/channel to 3.6 keV/channel and o_h to vary from -4.0 keV to 4.0 keV. The way we choose the number of channels in the principal components and the variation range for gain and offset guarantees the first half of equation <4.3b-2>:

$$n'_{min} \le m_{min} \qquad \text{<4.3b-3>}$$

However, the second half of equation <4.3b-2>:

$$m_{max} \le n'_{max} \qquad \text{<4.3b-4>}$$

is not guaranteed all the time. In case when $n'_{max} \le m_{max}$, we extrapolate the end of $h_0(x)$ so that $h_0(x)$ does not go to zero until channel 80. The extrapolation is linear as shown in Figure 4.11. For $x = n'_{max} + 1,\ n'_{max} + 2, \cdots, 80$

$$h_0(x) = \frac{80 - x}{80 - n'_{max}}\, h_0(n'_{max}) \qquad \text{< 4.3b-5 >}$$

at x = 80, $h_0(x)$ reaches zero.

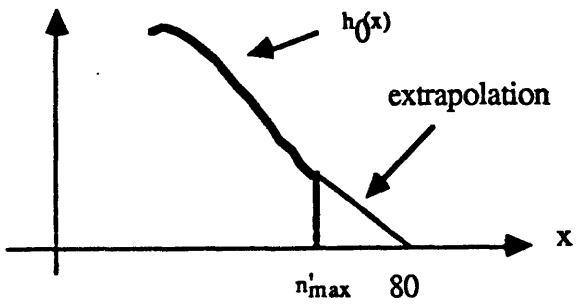

Figure 4.11 Extrapolation for $h_0(x)$

4.4 The Minimization Algorithm

In Section 4.3b, we define the variation range for g_h is from 2.8 keV/channel to 3.6 keV/channel, and for o_h is from -4.0 keV to 4.0 keV. To estimate g_h and o_h, we need to find gain and offset (g, o) that minimize the cost functions defined in Section 4.2. In the implementation, we first do a grid search to get a rough estimation. For gain estimation, compute the cost function from 2.5 keV/channel to 5.0 keV/channel for every 0.05 keV/channel increase in gain, and record the gain that minimizes the cost function. For offset estimation, compute the cost function from -6.0 keV to 6.0 keV for every 0.4 keV increase in offset, and record the offset that minimizes the cost function. We can also do a two dimensional grid search to get the estimation for gain and offset at the same time.

In the case when we only allow either gain or offset to vary, we can use Brent's method [10] to search for a better estimation. We modify the subroutine *brent* in Section 10.2 in Numerical Recipes in C [10]. The modified subroutine is called *brent1*, and is attached in the file "subrts.c" in Appendix 2.

Chapter 5 Performance Analysis

In the previous chapters, we developed our algorithms for estimating gain and offset of a PHS without using any radiation stabilization sources. In this chapter, we implement the least square error estimation and maximum likelihood estimation algorithms and show how the algorithms perform.

5.1 Preliminaries on Performance Analysis

5.1a The Data

The data used in our analysis here are from an ensemble acquired by a tool with a 6.25 inch L[14].. The ensemble consists of three scans:

1. sd133
2. sd133a
3. sd133c

Every scan listed above consists of 240 pulse height spectra of 94 channels[15] obtained with the standard gain ($g_0 = 3.2$ keV/channel) and offset ($o_0 = 0.0$ keV).

In the ensemble, the first scan sd133 was made when no material is inserted between the gamma ray device and the formation. The second scan sd133a was made when a sheet of 0.21 inch thick plexiglass was inserted. And the third scan sd133c was made when a sheet of 0.26 inch thick barite rubber was inserted.

5.1b Noise Adding

PHS obtained from well logging usually contain a high level of counting noise. The scans listed above were all obtained in the lab and the PHS in the scans consist of very little

---

[14] Please see the device in Figure 2.2. L is the distance between the gamma ray source and the scintillator detector.

[15] See Footnote 9 in Section 3.1. The first 5 channels in the original PHS acquired by Dr A. Becker at Schlumberger-Doll Research are excluded from our data files.

noise compared to the data obtained in well logging. The data obtained in the lab can be assumed noise free. We can model the field data by adding noise to the lab data and test the performance of our estimation algorithms on the noise added data.

In Section 4.2b, we modeled the noise with the Poisson distribution. Let y be a Poisson random variable. Let m_y be the mean of y and σ_y^2 be the variance of y, then σ_y is the standard deviation of y. The standard deviation is used to describe noise level. For the Poisson distribution, the mean and the variance are the same, that is $\sigma_y^2 = m_y$. Then the standard deviation of y is just $\sqrt{m_y}$. The larger the mean is, the larger the standard deviation will be.

The noise levels for field data are associated with logging speed. In our analysis, we use two logging speed:

$$v_1 = 600 \text{ ft / hr } = 2 \text{ in / sec} \qquad \text{<5.1b-1>}$$

and

$$v_2 = 1800 \text{ ft / hr } = 6 \text{ in / sec} \qquad \text{<5.1b-2>}$$

An average PHS obtained from well logging contains 5×10^5 counts/sec. Let $\{cf(x)\}$ be the average PHS obtained from well logging and TCf be the total number of counts/sec in $\{cf(x)\}$, then

$$TCf = \sum_x \{cf(x)\} = 5 \times 10^5 \text{ counts / sec} \qquad \text{<5.1b-3>}$$

We need to scale the PHS in the ensembles so that the average PHS is the same as the field data. Let c(x) be a PHS from one of the scans in the ensembles and $\{c(x)\}$ be the average spectrum for the scan. Note that c(x) is the counts per second. From Section 3.1, we know that c(x) is accumulated over 0.25 inch. Let v be the logging speed, then the time t used to accumulate c(x) is:

$$t = \frac{0.25 \text{ in}}{v \text{ in / sec}} = \frac{1}{4v} \text{ sec} \qquad \text{<5.1b-4>}$$

Let $\{c(x)\}$ be the average spectrum for c(x) and TC be the total number of counts/sec in $\{c(x)\}$,

$$TC = \sum_x |c(x)|$$

Let s be the scaling factor for c(x), then

$$s = \frac{TCf}{TC} t = \frac{5 \times 10^5 \text{ counts / sec}}{[\text{ TC counts / sec }]} \frac{1}{4v} \text{ sec} = \frac{1.25 \times 10^5}{TC\ v} \text{ sec} \qquad \text{<5.1b-4>}$$

Therefore, c(x)s contains counts of photons in each channel. Now, let us generate noisy data. For every channel x in c(x), let c(x)s be the mean, we use a Fortran subroutine to generate a number cp(x), which follows Poisson distribution with the mean c(x)s. Let $nc(x) = \frac{cp(x)}{s}$, then nc(x) is the noisy data corresponding to logging speed v, but scaled back to be comparable to a laboratory count rate spectrum. The nc(x) spectra are the noisy data used in our tests of algorithm performance.

Two noisy scans generated from scan sd133 and sd133a are used in the performance tests. One of the noisy scan, sd32, is generated by adding noise to every spectrum in sd133 corresponding to 600 ft/hr logging speed. For every spectrum c(x) in sd133, compute nc(x) corresponding to v = 600 ft/hr and put nc(x) to sd32. The other noisy scan, sd36a, is generated by adding noise to sd133a corresponding to 1800 ft/hr logging speed.

Recall the cost function for least square error estimation:

$$\varepsilon_L^2(g, o) = \frac{1}{N} \sum_{x=1}^{N} \frac{(\hat{z}(x) - h_0(x))^2}{\hat{z}(x)} \qquad \text{< 4.2c-1 >}$$

and the cost function for maximum likelihood estimation:

$$\varepsilon_{ML}^2(g, o) \equiv \frac{N}{2} \varepsilon_L^2(g, o) + \frac{1}{2} \sum_{x=1}^{N} \ln[2\pi \hat{z}(x)] \qquad \text{< 4.2c-2 >}$$

If $\hat{z}(x)$ and $h_o(x)$ are scaled by a constant s, $\varepsilon_L^2(g, o)$, the cost function for the least square error estimation is also scaled by s. Therefore, the scaling of $\hat{z}(x)$ and $h_o(x)$ does not change the cost function. However, the scaling of $\hat{z}(x)$ and $h_o(x)$ change the cost function for maximum likelihood estimation. In equation <4.2c-2>, when $\hat{z}(x)$ and $h_o(x)$ are scaled by s, the cost function becomes:

$$\varepsilon_{ML}^2(g, o, s) \cong \frac{N}{2}\varepsilon_L^2(g, o)s + \frac{1}{2}\sum_{x=1}^{N} \ln[2\pi s\hat{z}(x)] = \left[\frac{N}{2}\varepsilon_L^2(g, o)\right]s + \frac{1}{2}\sum_{x=1}^{N} \ln[2\pi\hat{z}(x)] + \frac{N\ln(s)}{2}$$

where the first term on the right side is also scaled by s, but the second term is just increased by a constant $\frac{N\ln(s)}{2}$. Constant terms in a cost function do not contribute to the estimation algorithm and therefore, can be dropped. If we drop the constant term $\frac{N\ln(s)}{2}$ in the equation above, the equivalent cost function is

$$\varepsilon_{ML}^2(g, o, s) \cong \frac{N}{2}\varepsilon_L^2(g, o)s + \frac{1}{2}\sum_{x=1}^{N} \ln[2\pi\hat{z}(x)]$$

which shows that the weight on the two terms in the cost function (<4.2c-2>) are changed when $\hat{z}(x)$ and $h_o(x)$ are scaled. Therefore, the results of ML estimation applied to the scaled data may be somewhat different from those obtained using the unscaled data. This difference has not been investigated.

When the noisy data are scaled back to the laboratory count rate, some of the estimation results from the noisy data can be more directly compared with the results from the noise free data.

5.1c Tests of Performance

The principal components for the ensemble are computed with equation <3.1> from all the PHS in scans sd133 and sd133c. As discussed in Section 4.3, the principal components are computed over 70 channels. Scan sd133 and sd133c and the two noisy scans generated from them, sd32 and sd36a, are used as testing scans.

Let c(x) be a PHS from one of the testing scans. Let g_h be a number randomly selected uniformly between 2.8 and 3.6, and o_h be a number randomly selected uniformly between -4.0 and 4.0. Let h(x) be the spectrum transformed from c(x) to (g_h, o_h). The gain and offset for h(x) are estimated to be (g, o) with the least square error estimation or maximum likelihood estimation algorithm discussed in Chapter 3.. In the following tests, h(x) is defined in different ways:

Test 1. Performance test on an entire scan with one variable
In this test, only one variable, either gain or offset is allowed to vary at a time.

Test 1a. Gain Estimation
For each spectrum c(x) in the scan, randomly select a g_h, define

$$h(x) = \frac{g_h}{g_0} c\left(\frac{g_h}{g_0} x\right)$$

and compute the estimated gain g.

Test 1b. Offset Estimation
For each spectrum c(x) in the scan, randomly select a o_h, define

$$h(x) = c\left(x + \frac{o_h - o_0}{g_0}\right)$$

and compute the estimated offset o.

Test 2. Performance test on a single spectrum with one variable
In this test, c(x) is a PHS selected from one testing scan, there are 20 different g_h and 50 different o_h.

Test 2a. Gain Estimation
For each selected g_h, define

$$h(x) = \frac{g_h}{g_0} c\left(\frac{g_h}{g_0} x\right)$$

and compute the estimated gain g.

Test 2b. Offset Estimation
For each selected o_h, define

$$h(x) = c\left(x + \frac{o_h - o_0}{g_0}\right)$$

and compute the estimated offset o.

Test 3. Performance test on a single spectrum with two variables
In this test, c(x) is a PHS selected from one testing scan, there are 30 different selected pairs of (g_h, o_h), For each pair of (g_h, o_h), define

$$h(x) = \frac{g_h}{g_0} c\left(\frac{g_h}{g_0} x + \frac{o_h - o_0}{g_0}\right)$$

and compute the estimated gain and offset (g, o).

The above three tests are performed on all the testing scans. For each of the three tests, the following performance measures are computed and plotted to show the performance of our estimation algorithm:

1. Estimation and the true value
   For Test 1 and Test 2, plot g_h vs g and o_h vs o. If the estimations are good, the two plots should form two lines around $g_h = g$ and $o_h = o$.
   For Test 3, plot $(g_h - g)$ vs $(o_h - o)$. If the estimations are good, we should find that all the points are around (0, 0).

2. Gain and offset deviations
   Compute the standard deviation for the estimated offset, and the relative standard deviation for the estimated gain.

$$\sigma_o = \left[(o - o_h)^2\right]^{\frac{1}{2}} \qquad \text{and} \qquad \sigma_g = \frac{\left[(g - g_h)^2\right]^{\frac{1}{2}}}{g_0}$$

   The deviations should be small for good estimations.

3. Chi-square
   Let $h_0(x)$ be the PHS transformed from h(x) with the estimated (g, o) to the standard gain and offset, and $\hat{z}(x)$ be the principal components

reconstruction of $h_0(x)$. Here the PHS $\hat{z}(x)$ is our estimate of the original spectrum c(x). Compute Chi-square

$$\frac{1}{N}\chi^2 = \frac{1}{N}\sum_x \frac{[\hat{z}(x) - c(x)]^2}{c(x)}$$

4. Window

Very often, "window" counts are used to estimate formation properties in well logging. A window is a partial integral over the spectrum. For c(x), the window is defined as

$$w = \sum_x c(x)$$

where x includes all the channels with energy between 150 keV and 250 keV. The estimated window $\hat{w}$ is computed from $\hat{z}(x)$:

$$\hat{w} = \sum_x \hat{z}(x)$$

We plot the estimated window $\hat{w}$ vs. the original window w. The plot should form a line around $\hat{w} = w$.

5.2 Performance on scan sd133 and sd32

The scan sd133 and sd133c are used to compute the principal components for the ensemble. In this section, we perform the three tests discussed in Section 5.1 on scan sd133. In Section 5.3, we perform the same tests on scan sd133a which is not used for computing the principal components. If our estimation algorithms perform well on both sd133 and sd133a, we can expect the algorithms to be useful.

In this section and Section 5.3, we perform the tests on the noise free data, and data with noise added corresponding to a 600 ft/hr logging speed.

5.2a Tests on Noise Free Data

We use scan sd133 for all the tests in this section.

5.2a-1 Performance Test on Gain Estimation over the Entire Scan

Table 5.2a-1 and all the figures in this subsection are resulted from Test 1a on sd133. Test 1a is the performance test for gain estimation over the entire scan sd133 as defined in Section 5.1c.

Table 5.2a-1 Relative Standard Deviation for Gain Estimated over sd133

	Reconstruction with Projection Coefficients[16]	Reconstruction with Optimal Coefficients[17]
LSE[18]	σ_g = 0.0109	σ_g = 0.0111
ML[19]	σ_g = 0.2422	σ_g = 0.2705
BRENT[20]	σ_g = 0.0107	σ_g = 0.0108

The relative standard deviation for the estimated gain is the estimation error as defined in Section 5.1c. Table 5.2a-1 shows six estimation errors for gain estimated with different reconstruction coefficient vectors, different cost functions and different cost function minimization methods. The six estimation errors are listed in two columns and three rows. The estimation errors in the first and the third row are all around 0.01, which means the estimation error is only around one percent. The estimations are very good, as we will see in the figures in this section.

---

16 Defined in equation <4.2a-6> in Section 4.2a

17 Defined in equation <4.2b-18> in Section 4.2b

18 Cost function for least square error estimation defined in equation <4.2a-8> in Section 4.2a is minimized by grid search method discussed in 4.4.

19 Cost function for maximum likelihood estimation defined in equation <4.2c-2> in Section 4.2b is minimized by grid search method.

20 A cost function minimization method discussed in Section 4.4. The cost function for LSE estimation is used.

All the estimation errors in the first column of Table 5.2a-1 are obtained when the projection coefficient vector are used to compute the cost functions. All the estimation errors in the second column are obtained when the optimal coefficient vector are used to compute the cost functions. The projection coefficient vector is defined in Section 4.2a, and the optimal coefficient vector is defined in Section 4.2b. In the first row (LSE), the cost function used for the estimation is the defined in equation <4.2a-8> for least square estimation, and the cost function is minimized with the grid search method discussed in Section 4.4. In the second row (ML), the cost function used is defined in equation <4.2c-2> and minimized with the grid search method. In the third row (BRENT), the cost function used is the same least square error estimation as in the first row, and the cost function is minimized with Brent's method by starting from the result of the grid search method.

Compare the three rows in each column, we expect that the third row to be an improvement for the first row, which is actually the case. The third row shows smaller estimation errors. We also expect that the second row to be an improvement for the first row. However, this is not the case. The estimation error in the second row is much larger than in the other two rows. The maximum likelihood estimation algorithm is not quite as good as we expected it to be. The reason for this discrepancy is not known.

Compare the two columns in each row, the estimation with the optimal coefficient vector should be better than the estimation with the projection coefficient vector, as discussed in Section 4.2b. However, the estimation errors for gain estimated with the optimal coefficient vector in the second column are bigger than the corresponding ones in the first column. The actual performance with the optimal coefficient vector is worse than the projection coefficient vector. In the offset estimation and the estimations for other scans, we will see the same kind of performance. Why the optimal coefficient vector does not give a better estimation and what will be the best way to define the reconstruction coefficient vector is left for further investigation.

In this chapter, all the figures are plots of the estimation results obtained using the projection coefficient vector. The following figures give further details of the performance of the algorithms.

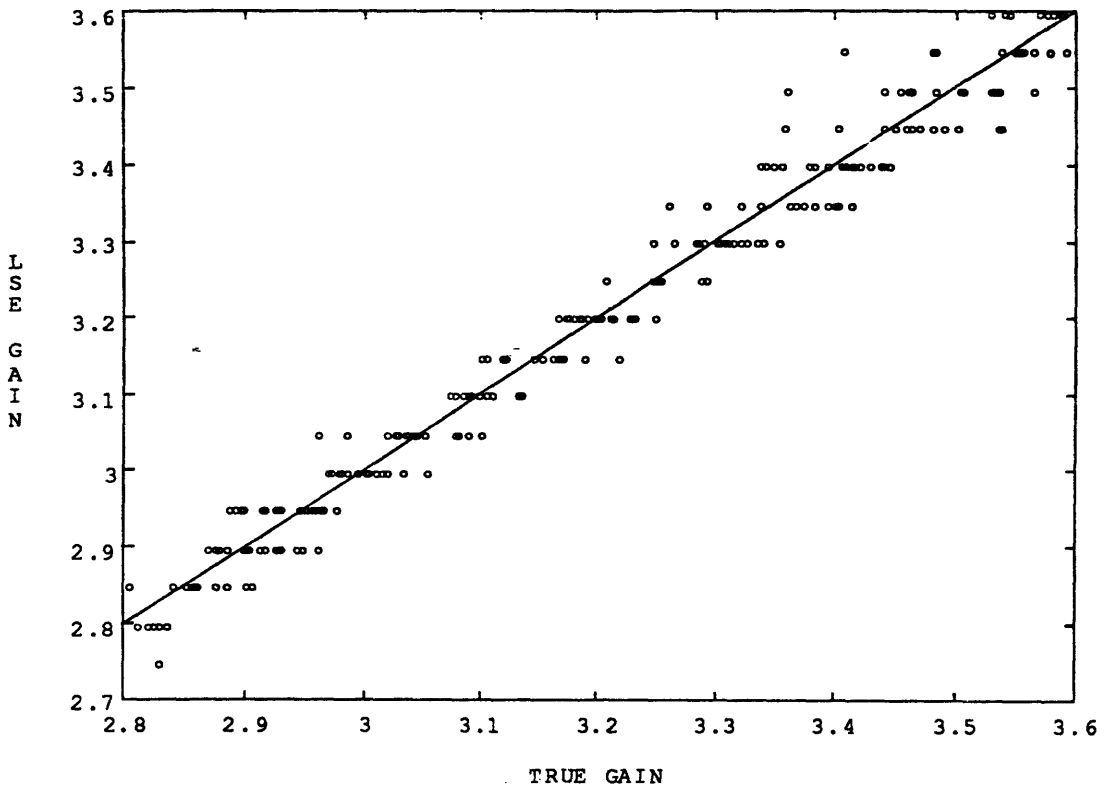

Figure 5.2a-1: Plot of LSE estimation for gain vs. the actual gain of the testing PHS in scan sd133. The estimated gains are discrete because grid search method is used for cost function minimization. The estimation is good as all the points plotted are around the line where the estimated gain is equal to the actual gain. The relative standard deviation , or the estimation error is 0.0109.

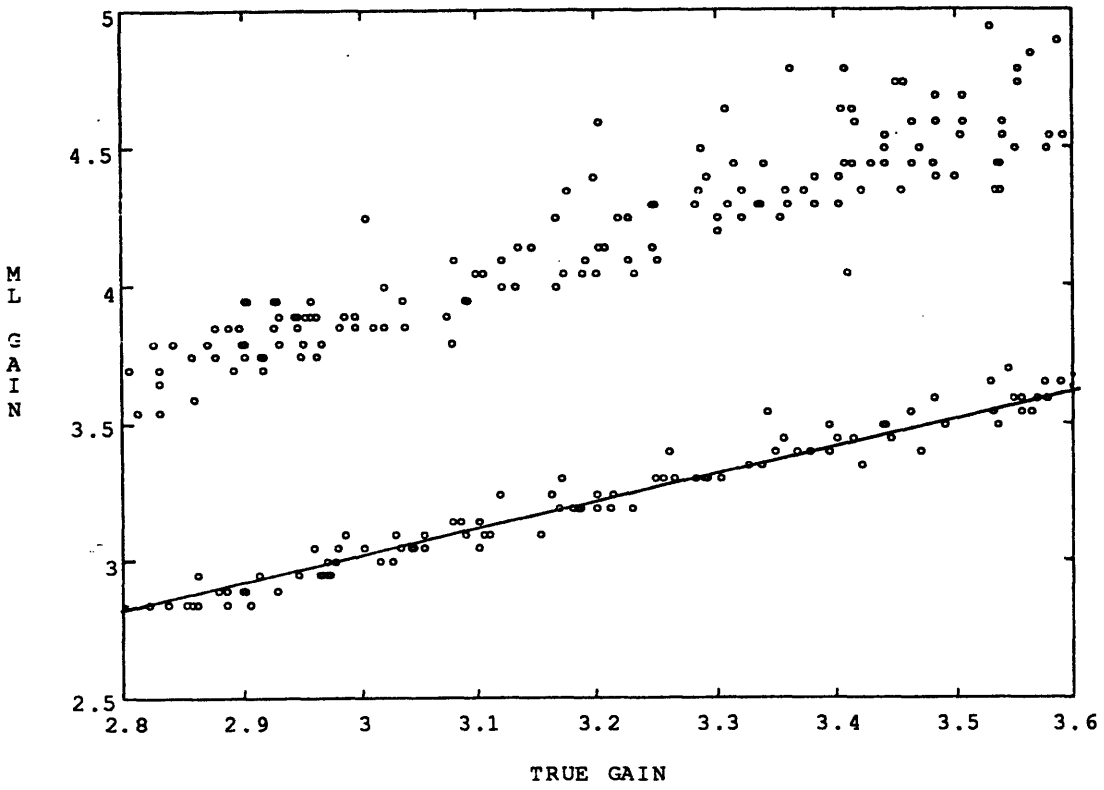

Figure 5.2a-2: Plot of ML estimation for gain vs. the actual gain of the testing PHS in scan sd133. The estimated gains should be discrete because grid search method is used for cost function minimization. However, the discrete nature of the estimated gain is not as clear as in Figure 5.2a-1 because the estimated gain here is more spread out. Part of the above estimation is good as some of the points plotted are around the line where the estimated gain is equal to the actual gain. Many points in the plot are above the line. The relative standard deviation is 0.2422, much bigger than the LES estimation. This bimodal distribution of the estimated gains indicates a problem with the ML estimation algorithm which is not understood.

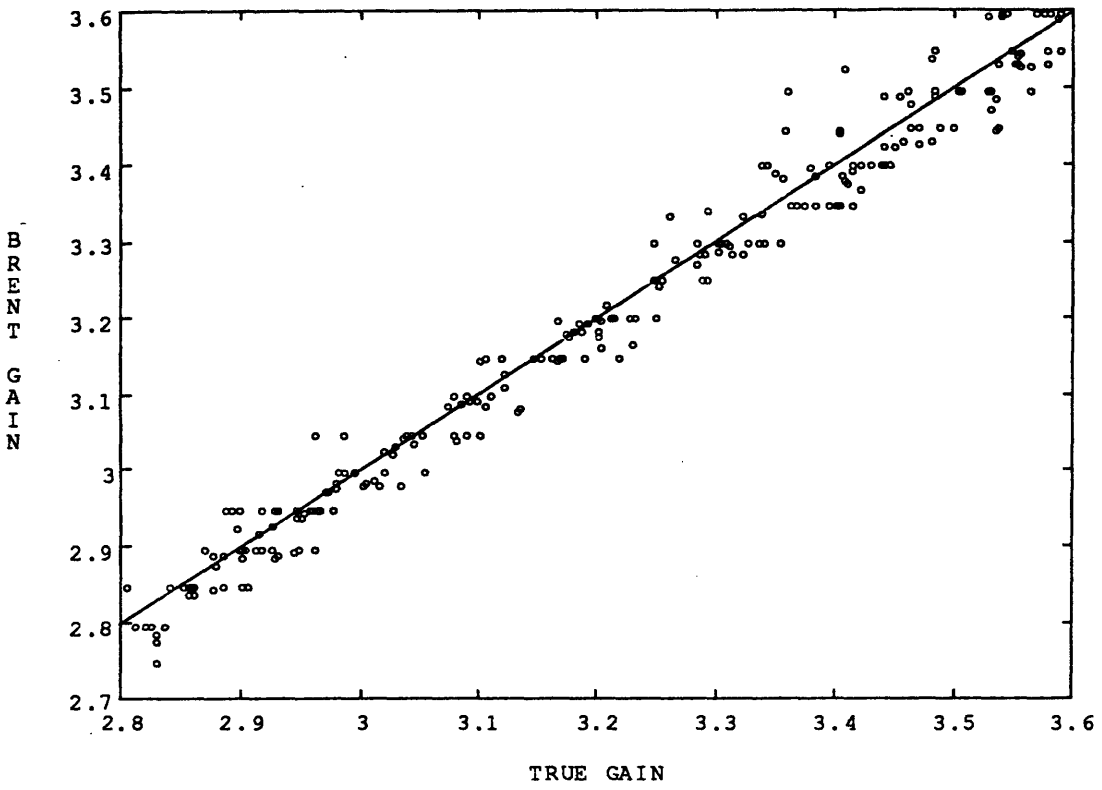

Figure 5.2a-3: Plot of LSE estimation for gain vs. the actual gain of the testing PHS in scan sd133. The estimated gains are closer to the actual gains than in Figure 5.2a-1 because Brent's method is used for cost function minimization. The estimation is good as all the points plotted are around the line where the estimated gain is equal to the actual gain. The relative standard deviation , or the estimation error is only 0.0107.

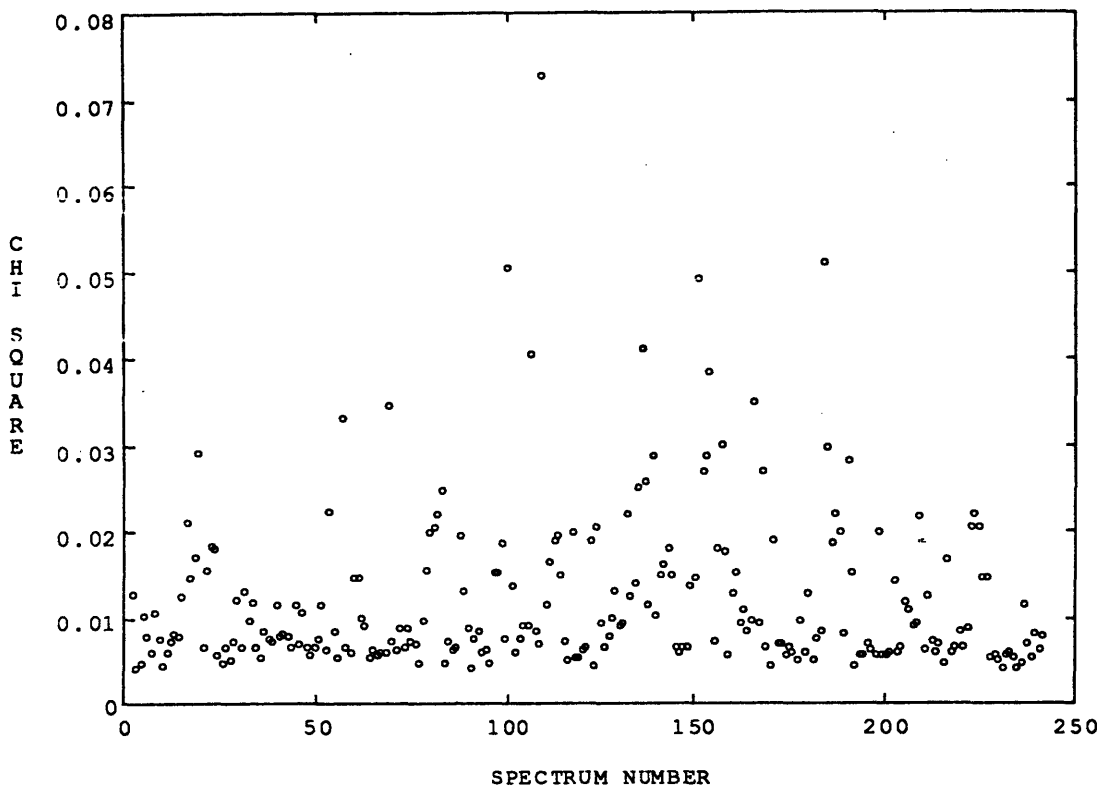

Figure 5.2a-4: Plot of Chi-square.

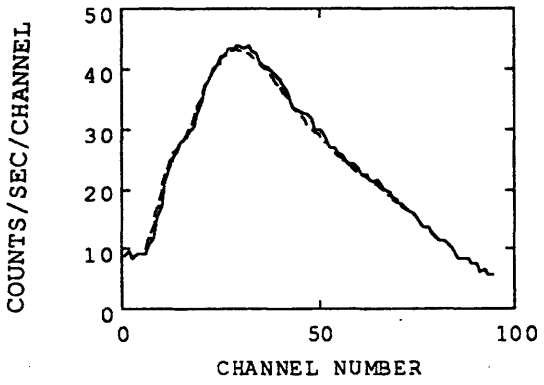

Figure 5.2a-5: The solid curve represents the 100th spectrum c(x) in sd133 and the dashed curve is the estimated spectrum $\hat{z}(x)$. Chi-square is 0.012.

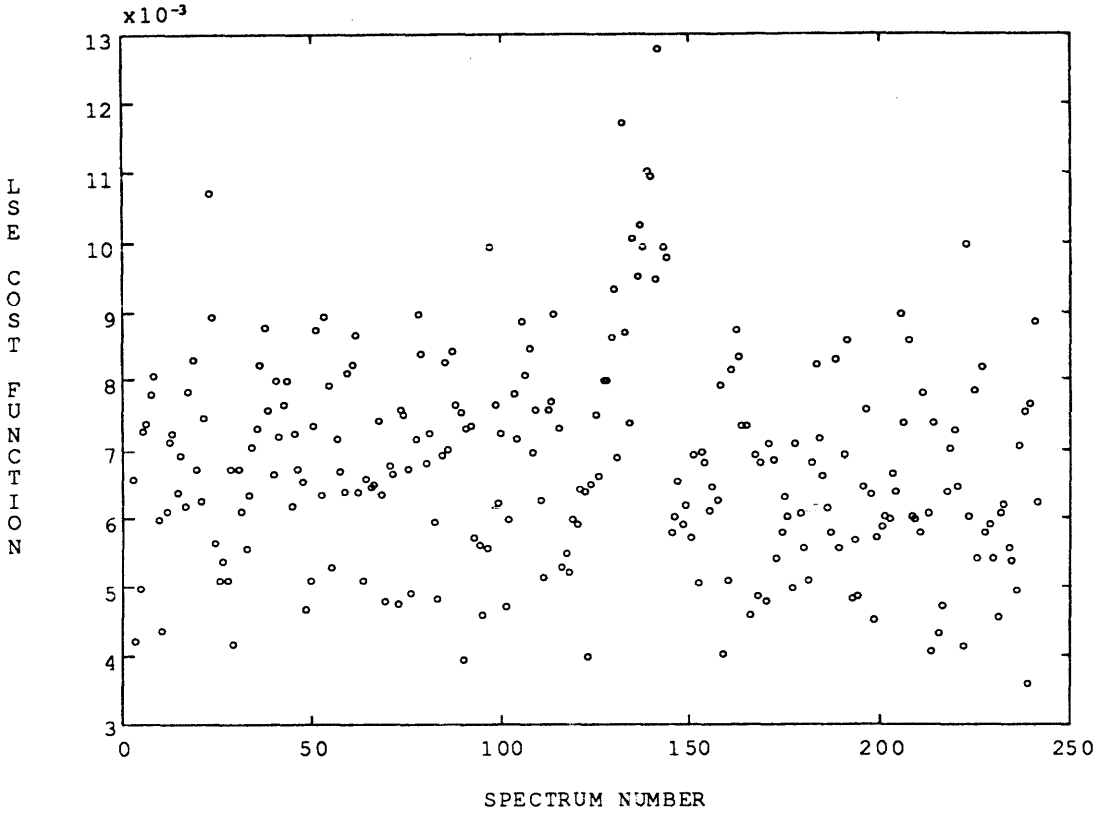

Figure 5.2a-6: The minimum of LSE cost function resulted from Brent's method for the estimation on every spectrum in sd133.

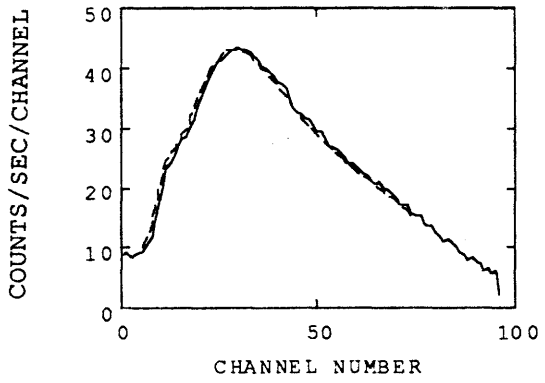

Figure 5.2a-7: The solid curve represents the 100th spectrum $h_0(x)$ in sd133 and the dashed curve is the transformed spectrum $\hat{z}(x)$. The minimum LSE cost function is 0.005.

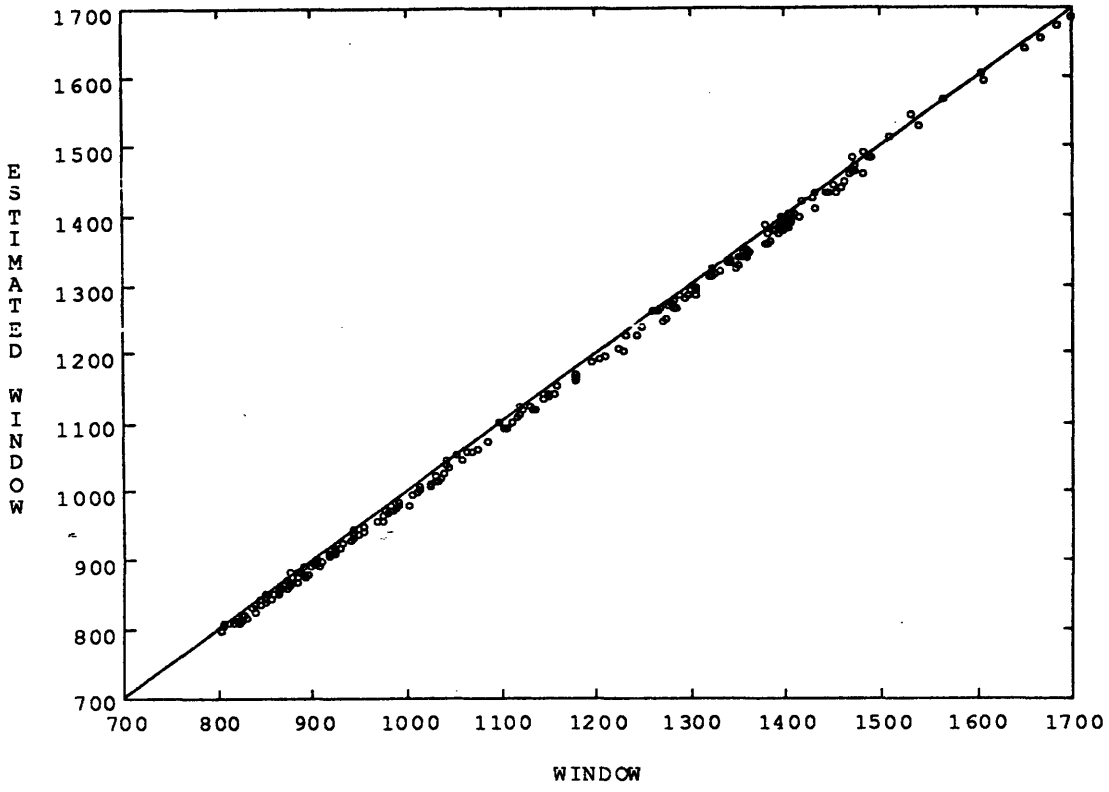

Figure 5.2a-8 Plot of estimated window $\hat{w}$ vs. the original window w. The plot forms a straight line around $\hat{w} = w$. The dispersion of the estimates is reasonably low, but they appear to be slightly biased.

5.2a-2 Performance Test on Offset Estimation over the Entire Scan

Table 5.2a-2 and all the figures in this subsection are resulted from Test 1b on scan sd133. Test 1b is the performance test for offset estimation over the entire scan sd133 as defined in Section 5.1c.

Table 5.2a-2 Standard Deviation for Offset Estimation

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_o = 0.5968	σ_o = 0.6029
ML	σ_o = 2.8778	σ_o = 2.6870
BRENT	σ_o = 0.5901	σ_o = 0.5987

The standard deviation for the estimated offset is the estimation error as defined in Section 5.1c. The standard deviations in the above table are listed in the similar manner as the relative standard deviations listed in Table 5.2a-1. The least square error estimation has very small estimation errors which is about 0.6 keV.

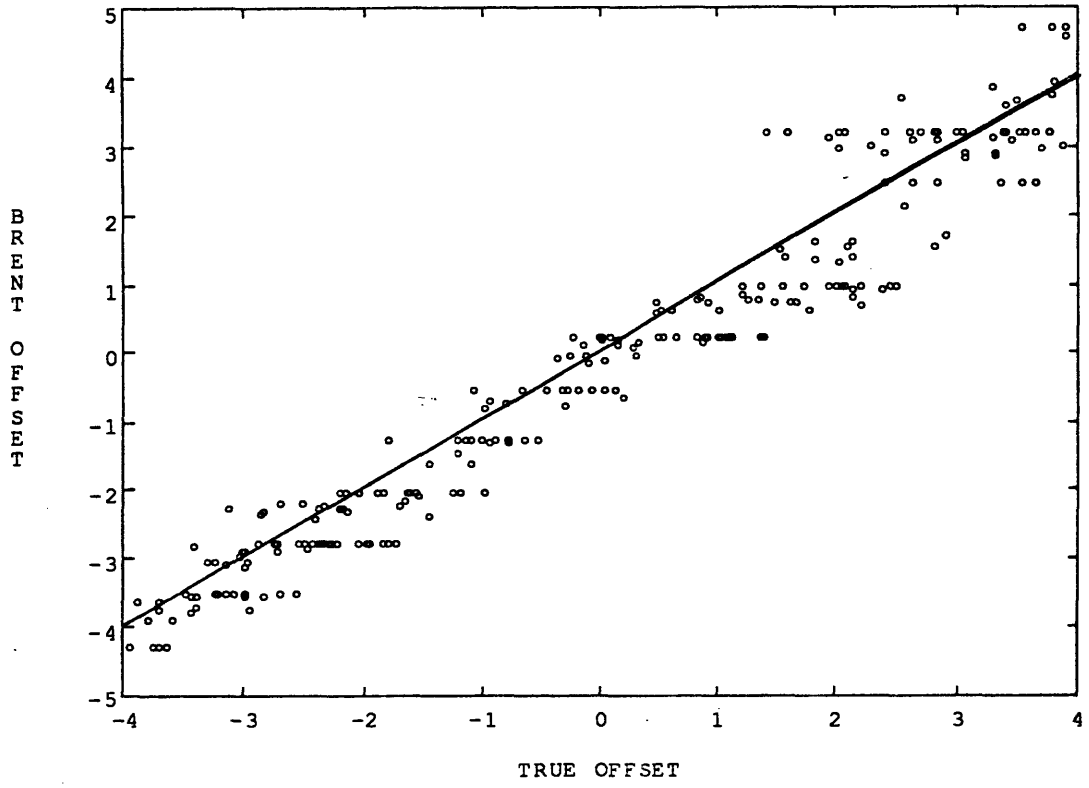

Figure 5.2a-9 Plot of BRENT estimation for offset vs. the actual offset of the testing PHS in scan sd133. The estimation is good as all the most of the points plotted are around the line where the estimated offset is equal to the actual offset even though some bias is apparent. The standard deviation is 0.1844.

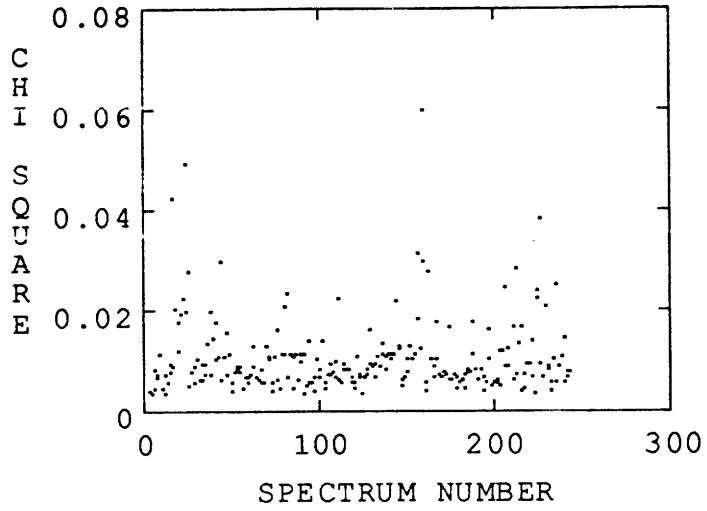

Figure 5.2a-10 Plot of Chi-square

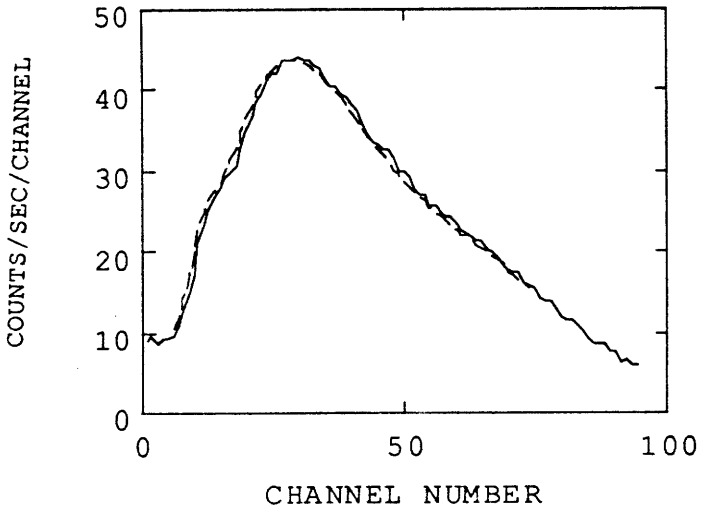

Figure 5.2a-11: The solid curve represents the 100th spectrum c(x) in sd133 and the dashed curve is the estimated spectrum $\hat{z}(x)$. Chi-square is 0.0054.

5.2a-3 Performance Test on Gain Estimation on a Single Spectrum

Table 5.2a-3 and all the figures in this subsection are resulted from Test 2a on the fiftieth spectrum in sd133. Test 2a is the performance test for gain estimation on a single spectrum.

Table 5.2a-3 Relative Standard Deviation for Estimated Gain

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_g = 0.0142	σ_g = 0.0142
ML	σ_g = 0.3518	σ_g = 0.3574
BRENT	σ_g = 0.0115	σ_g = 0.0118

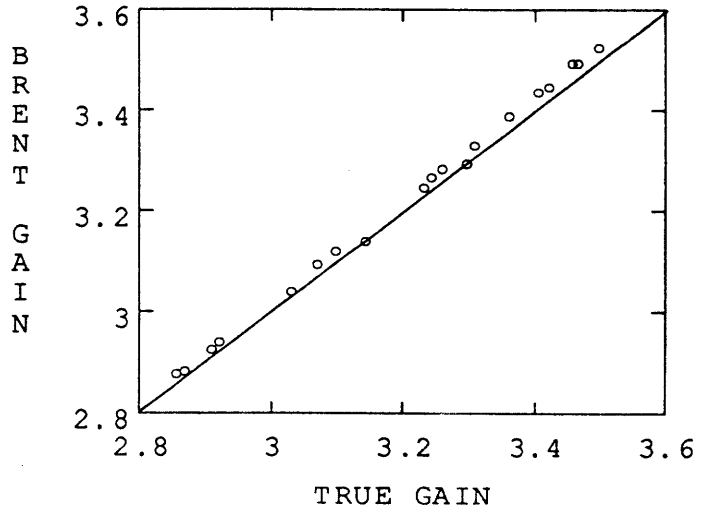

Figure 5.2a-12 Estimated gain vs. the actual gain. Again, the dispersion is low, but some bias in the estimated gain is apparent.

5.2a-4 Performance Test on Offset Estimation on a Single Spectrum

Table 5.2a-4 and all the figures in this subsection are resulted from Test 2b on the 100th spectrum in sd133. Test 2b is the performance test for offset estimation on a single spectrum.

Table 5.2a-4 Standard Deviation for Estimated Offset

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_o = 0.3696	σ_o = 0.3696
ML	σ_o = 2.4684	σ_o = 2.5117
BRENT	σ_o = 0.3440	σ_o = 0.3379

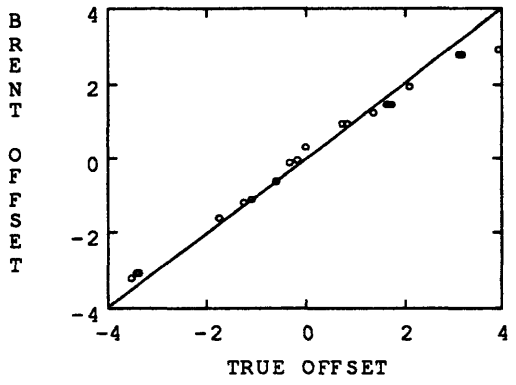

Figure 5.2a-13 Estimated offset vs. the actual offset

5.2a-5 Performance Test on a Single Spectrum with two Variables

Test 3 is the performance test for two dimensional estimation for gain and offset on a single spectrum as described in Section 5.1c. In this subsection, Test 3 is performed on the 100th spectrum in sd133. The projection coefficient vectors are used as the reconstruction coefficient vectors. Figure 14 shows the plot of the gain deviation vs. the offset deviation. All the points plotted are around the origin where both deviations are

zeros. The relative standard deviation for the estimated gain is 0.0119, and the standard deviation for the estimated offset is 0.6157 keV. The estimation error for both gain and offset are small. Therefore, our least square error estimation can be used for two dimensional estimation for gain and offset.

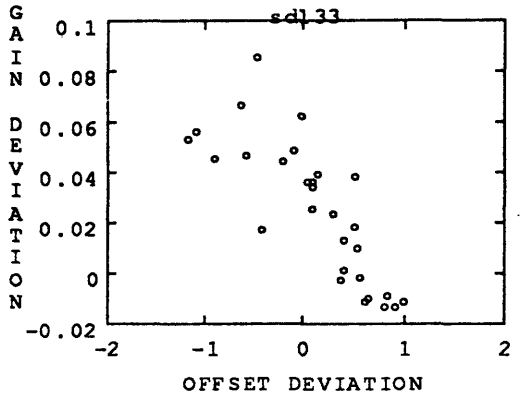

Figure 5.2a-14 Gain deviation vs. offset deviation

We have completed the performance tests on the noise free scan sd133. The estimation errors for the estimated gain in all the tests are close to one percent and the standard deviations for the estimated offset are close to 0.2 keV.

5.2b Noise Added at 600 ft/hr Logging Speed

The scan we used in this section is called sd32 which is generated by adding noise with logging speed at 600 ft/hr to scan sd133 as discussed in Section 5.1b. In this section, we do some of the performance tests on sd32 as we did on sd133 in Section 5.2a.

5.2b-1 Performance Test on Gain Estimation over the Entire Scan

Table 5.2b-1 and all the figures in this subsection are resulted from Test 1a on sd32. Test 1a is the performance test for gain estimation over the entire scan sd32 as defined in Section 5.1c.

Table 5.2b-1 Relative Standard Deviation for Gain Estimated over sd32

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_g = 0.0426	σ_g = 0.0429
ML	σ_g = 0.2898	σ_g = 0.3152
BRENT	σ_g = 0.0416	σ_g = 0.0419

Compared with Table 5.2a-1, the LSE estimation errors are about four times bigger in Table 5.2b-1. With noisy PHS, the estimation error for gain is four percent.

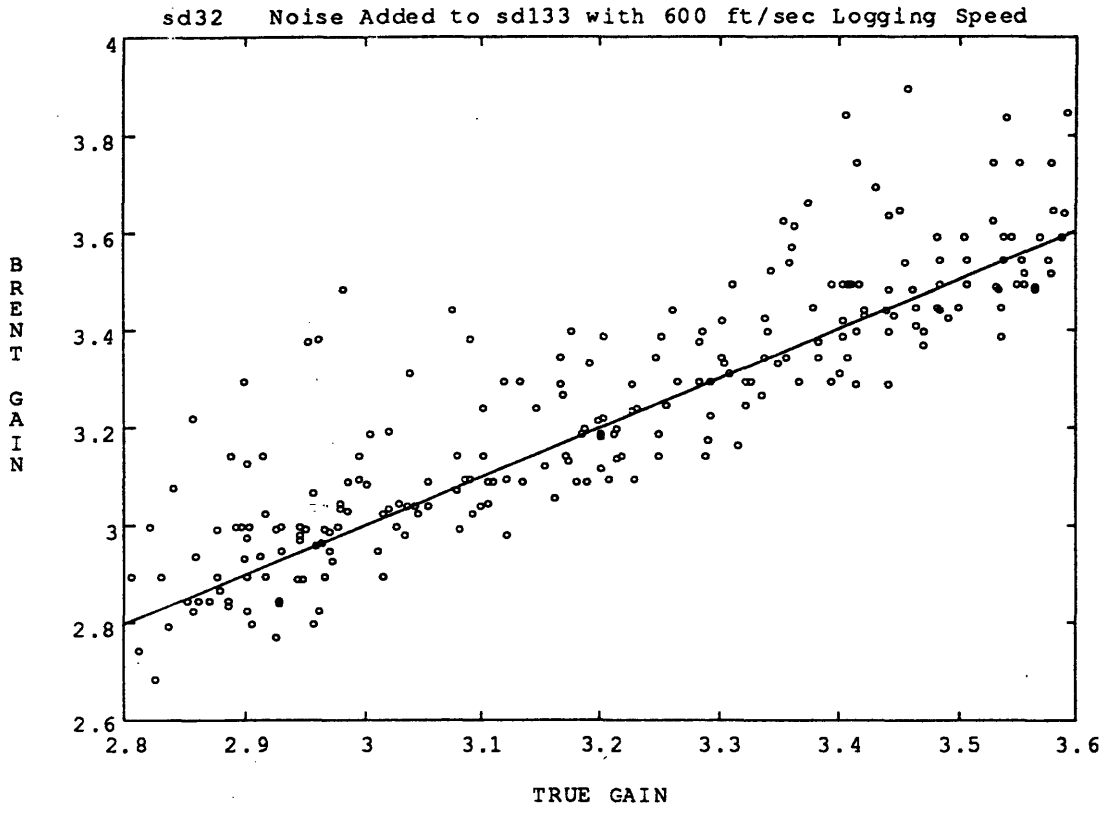

Figure 5.2b-1 Plot of LSE estimation for gain vs. the actual gain of the testing PHS in scan sd32. The estimation error is 0.0416.

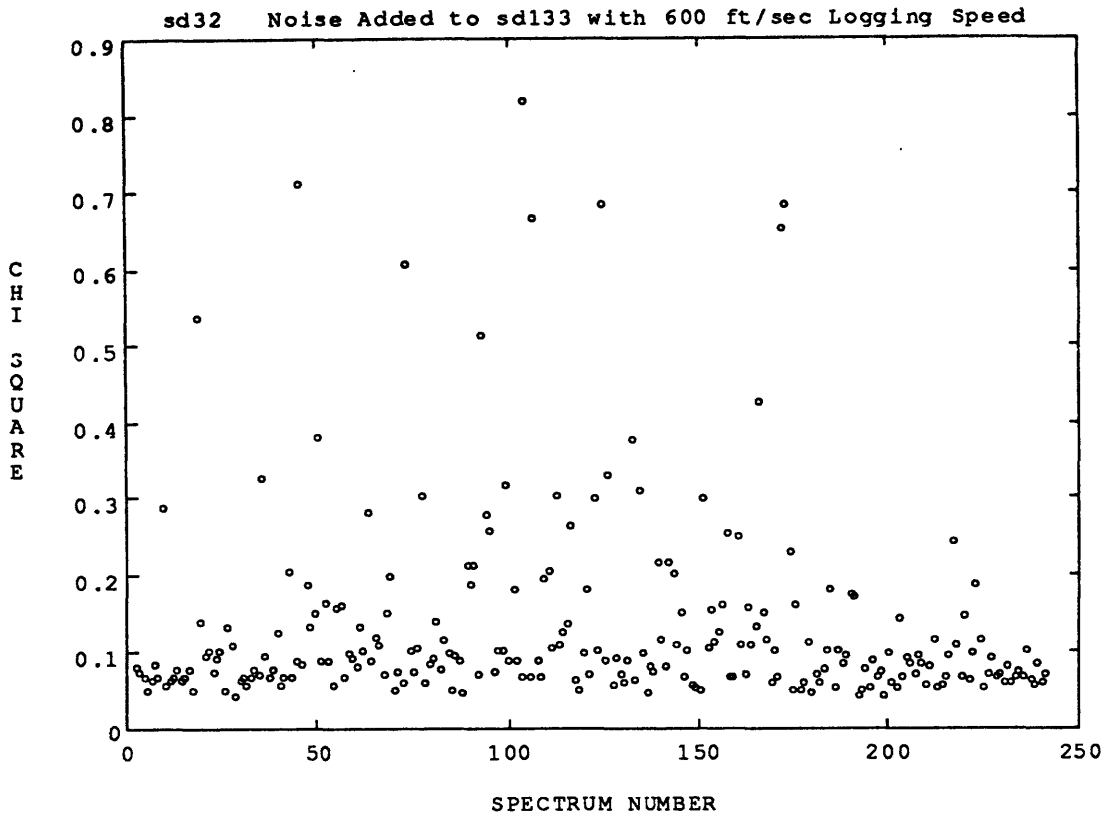

Figure 5.2b-2 Plot of Chi-square

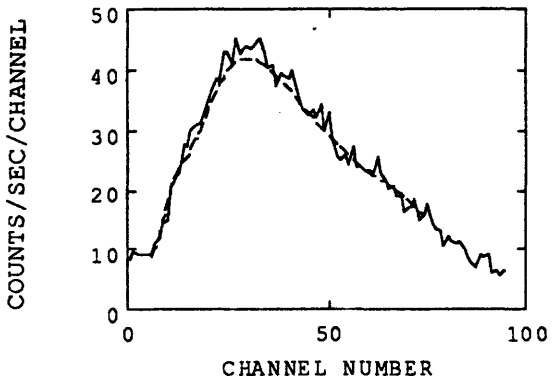

Figure 5.2b-3 The solid curve represents the 100th spectrum c(x) in sd32 and the dashed curve is the estimated spectrum for c(x). Here, c(x) is the nc(x) computed from the 100th spectrum in sd133. Chi square is 0.1573. Note that the estimated PHS is much smoother than c(x).

5.2b-2 Performance Test on Offset Estimation over the Entire Scan

Table 5.2b-2 and the figures in this subsection are resulted from Test 1b on scan sd32. Test 1b is the performance test for offset estimation over the entire scan sd32 as defined in Section 5.1c.

Table 5.2b-2 Standard Deviation for Offset Estimation over the Entire Scan

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_o = 1.6957	σ_o = 1.6979
ML	σ_o = 2.7670	σ_o = 2.6416
BRENT	σ_o = 1.7341	σ_o = 1.7315

Note that the standard deviation in the third row is bigger than the ones in the first row. With noisy data, the Brent's method does not necessarily improve the estimation from the grid search method.

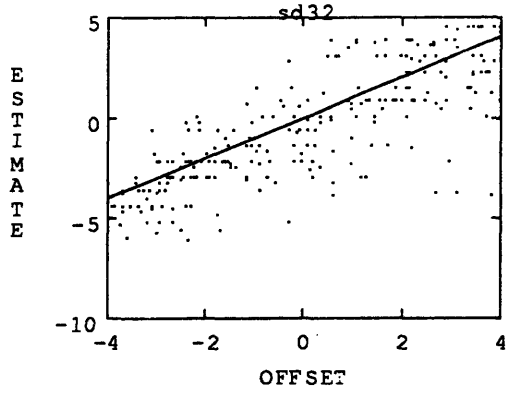

Figure 5.2b-4 Plot of LSE estimation for offset vs. the actual offset of the testing PHS in scan sd32. The estimation error is 1.7341.

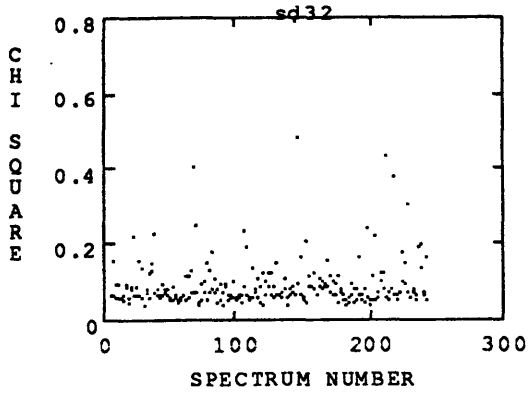

Figure 5.2b-5 Plot of Chi-square

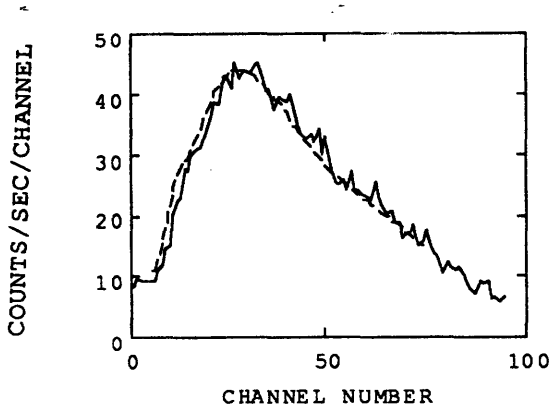

Figure 5.2b-6 The solid curve represents the 100th spectrum c(x) in sd32 and the dashed curve is the estimated spectrum for c(x). Chi square is 0.09. Note that the estimated PHS is smoother.

5.2b-3 Performance Test on Gain Estimation on One Spectrum

Table 5.2b-3 and the figure in this subsection are resulted from Test 2a on sd32. Test 2a is the performance test for gain estimation on one single spectrum in scan sd32 as defined in Section 5.1c.

Table 5.2b-3 Relative Standard Deviation for the Estimated Gain

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_g = 0.0162	σ_g = 0.0186
ML	σ_g = 0.3894	σ_g = 3989
BRENT	σ_g = 0.0144	σ_g = 0157

The relative standard deviation for gain in the LSE and BRENT estimations are around one and a half percent. Compare with Table 5.2b-1, the estimation error is much smaller.

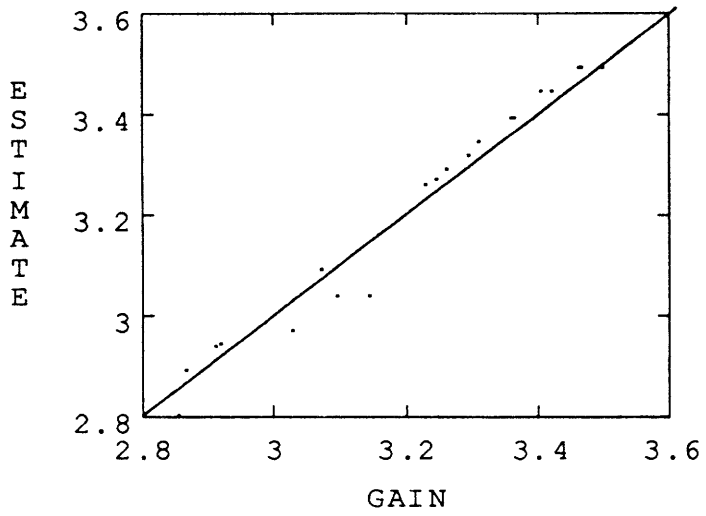

Figure 5.2b-7 Plot of BRENT estimation for gain vs. the actual gain of the testing PHS in scan sd32.

5.2b-4 Performance Test on Offset Estimation over One Spectrum

Table 5.2b-4 and the figure in this subsection are resulted from Test 2b on scan sd32. Test 2b is the performance test for offset estimation on one spectrum in scan sd32 as defined in Section 5.1c.

Table 5.2b-4 Standard Deviation for Offset Estimation on one Spectrum

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_o = 1.6916	σ_o = 1.6669
ML	σ_o = 2.7846	σ_o = 2.9885
BRENT	σ_o = 1.7183	σ_o = 1.7011

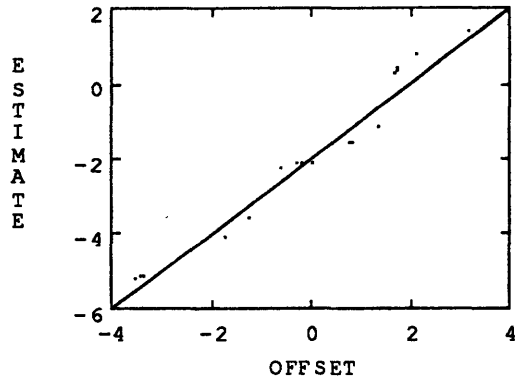

Figure 5.2b-1 Plot of LSE estimation for offset vs. the actual offset of the testing PHS in scan sd32. Note that the error is dominated by the bias in the estimation.

5.3 Performance on scan sd133a and sd36a

The principal components for the data ensemble are computed from scan sd133 and sd133c. Scan sd133a is not used to determine the principal components. In this section, we perform test 1 on sd133a and the noisy data generated from sd36a, which tests the robustness of our estimation algorithms.

5.3a Tests on Noise Free Data

5.3a-1 Performance Test on Gain Estimation over Scan sd133a

Table 5.3a-1 and the figures in this subsection are resulted from Test 1a on scan sd133a. Test 1a is the performance test for offset estimation on one spectrum in scan sd133a as defined in Section 5.1c.

Table 5.3a-1 Relative Standard Deviation for Gain Estimation

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_g = 0.0175	σ_g = 0.0170
ML	σ_g = 0.2341	σ_g = 0.2554
BRENT	σ_g = 0.0190	σ_g = 0.0185

Compare with Table 5.2a-1, the estimation errors for gain performed on sd133a are just slightly bigger than the estimation errors for sd133.

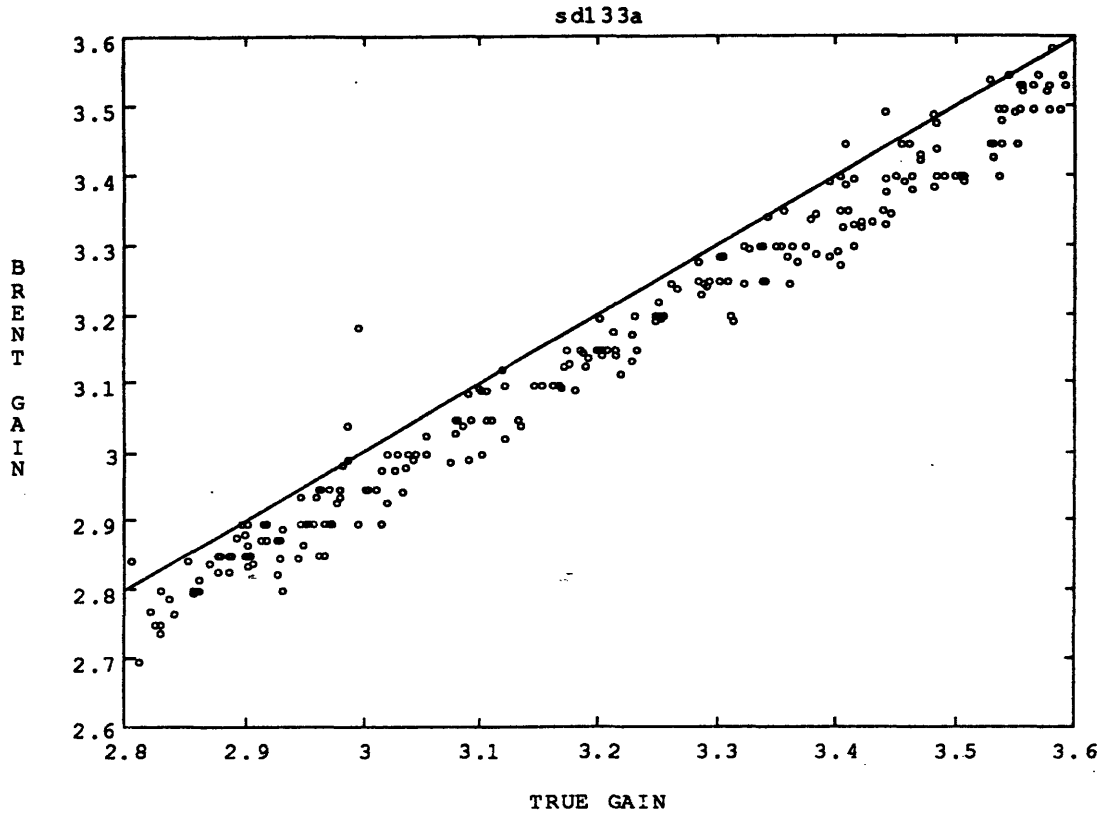

Figure 5.3a-1 Plot of estimated gain vs. the actual gain
Compared with sd133, the estimated gains for sd133a are biased.

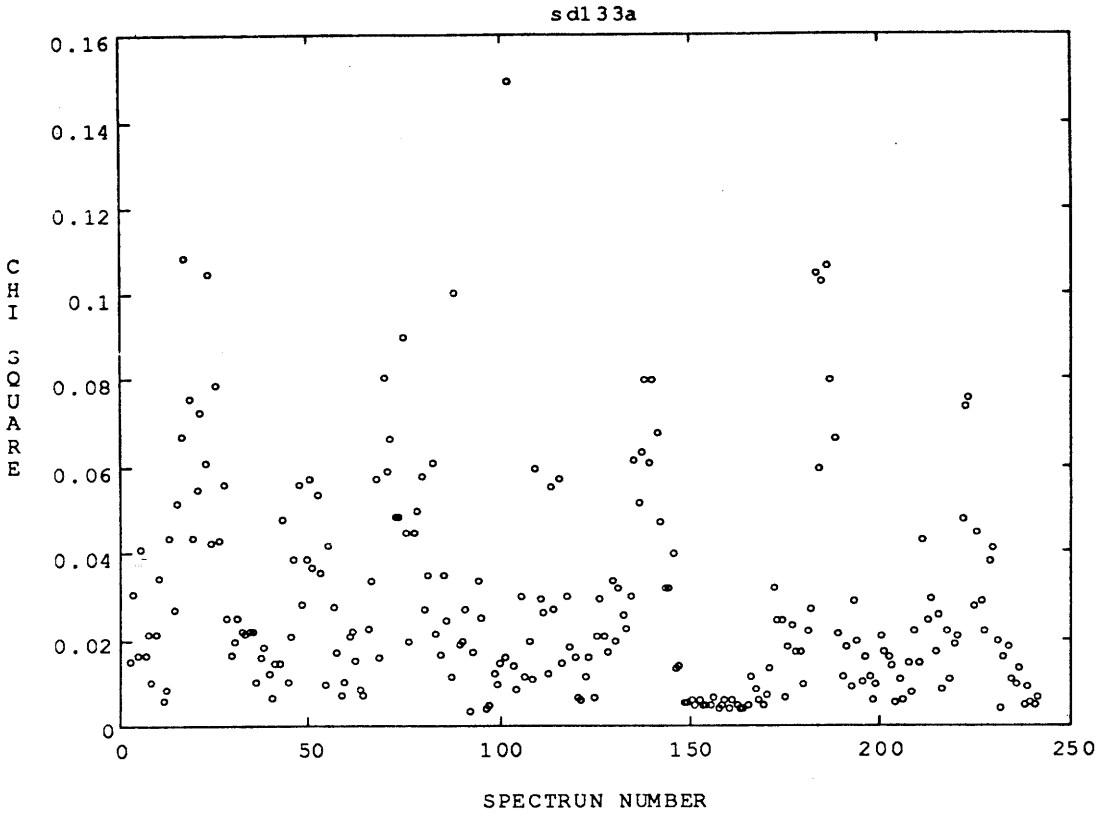

Figure 5.3a-2 Plot of Chi-square

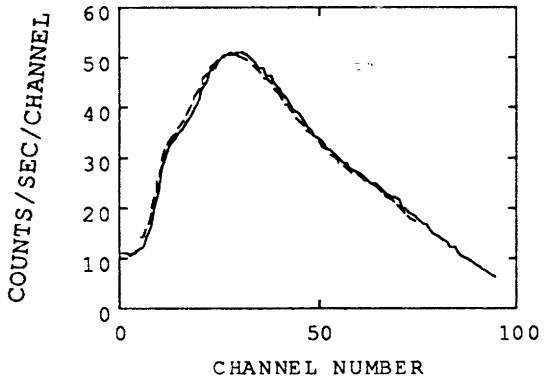

Figure 5.3a-3 The solid curve represents the 100th spectrum c(x) in sd133a and the dashed curve is the estimated spectrum for c(x). Chi-square is 0.0059.

5.3a-2 Performance Test on Offset Estimation over the Entire Scan

Table 5.3a-2 and the figures in this subsection are resulted from Test 1b on scan sd133a. Test 1b is the performance test for offset estimation over the entire scan sd133a as defined in Section 5.1c.

Table 5.2a-2 Standard Deviation for Offset Estimation over the Entire Scan

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_o = 0.6192	σ_o = 0.6073
ML	σ_o = 3.0454	σ_o = 2.9110
BRENT	σ_o = 0.5949	σ_o = 0.5846

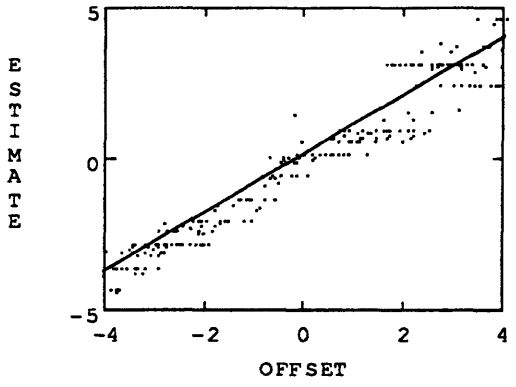

Figure 5.3a-4 Plot of BRENT estimated offset vs. the actual offset.

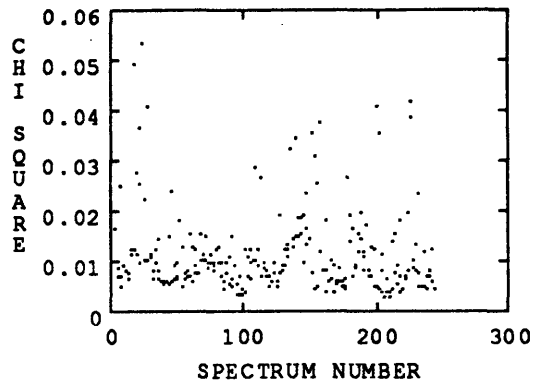

Figure 5.3a-5 Plot of Chi-square

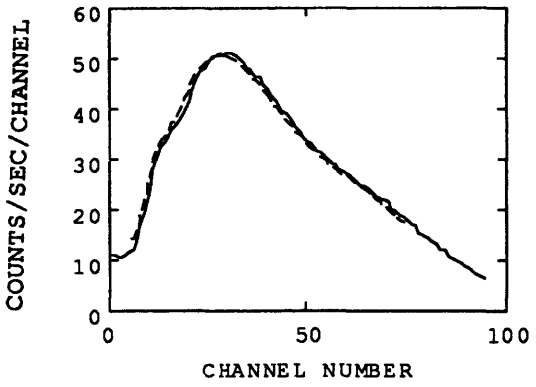

Figure 5.3a-6 The solid curve represents the 100th spectrum c(x) in sd133a and the dashed curve is the estimated spectrum for c(x). Chi-square is 0.0061.

5.3b Noise Added at 1800 ft / hr Logging Speed

The scan we used in this section is called sd36a which is generated by adding noise to sd133a corresponding to 1800 ft/hr logging speed. Scan sd36a is generated from the scan that is not used to determine the principal components and a high level noise is added. This is the most challenging test of our estimation algorithm.

5.3b-1 Performance Test on Gain Estimation over the Entire Scan

Table 5.3b-1 and all the figures in this subsection are resulted from Test 1a on sd36a. Test 1a is the performance test for gain estimation over the entire scan sd36a as defined in Section 5.1c.

Table 5.3b-1 Relative Standard Deviation for Gain Estimated over sd36a

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_g = 0.1276	σ_g = 0.1379
ML	σ_g = 0.3342	σ_g = 0.3561
BRENT	σ_g = 0.1264	σ_g = 0.1357

Note the estimation error on scan sd36a is much larger than the other scans.

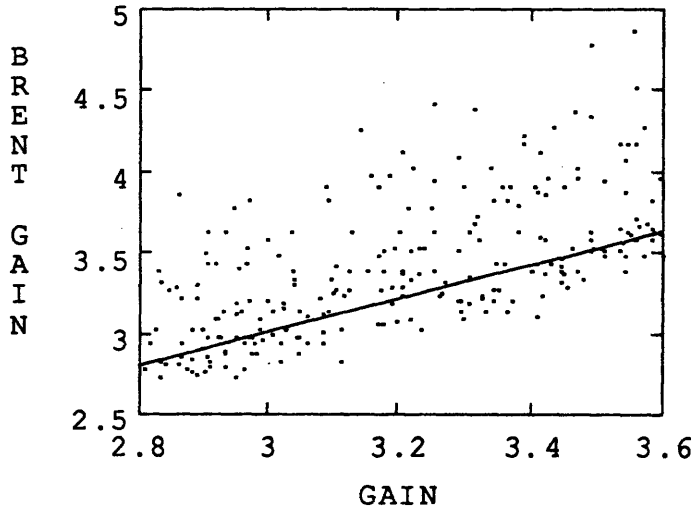

Figure 5.3b-2 Plot of estimated gain vs. the actual gain

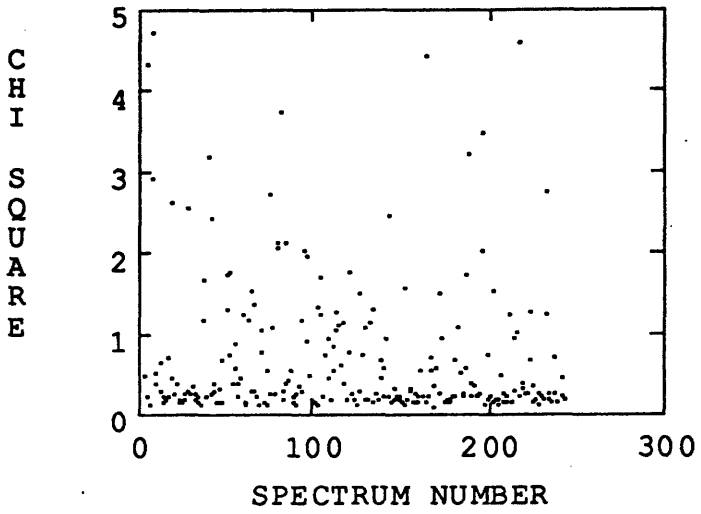

Figure 5.3b-2 Plot of Chi-square

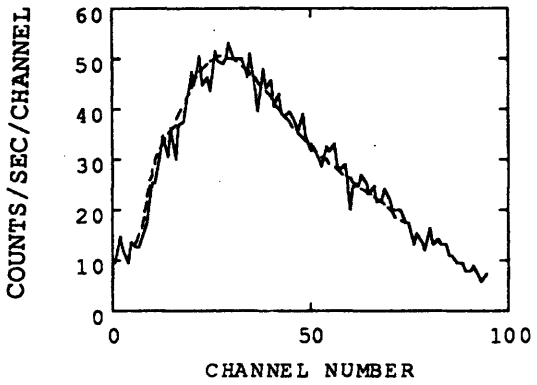

Figure 5.3b-3 The solid curve represents the 100th spectrum c(x) in sd36a and the dashed curve is the estimated spectrum for c(x). Chi-square is 0.1665.

5.3b-2 Performance Test on Offset Estimation over the Entire Scan

Table 5.3b-2 and all the figures in this subsection are resulted from Test 1b on scan sd36a. Test 1b is the performance test for offset estimation over the entire scan sd36a as defined in Section 5.1c.

Table 5.3b-2 Standard Deviation for Offset Estimation over the Entire Scan

	Reconstruction with Projection Coefficients	Reconstruction with Optimal Coefficients
LSE	σ_o = 2.3806	σ_o = 2.3880
ML	σ_o = 3.2119	σ_o = 3.1282
BRENT	σ_o = 2.4121	σ_o = 2.4238

Note the estimation error on the noisy scan is much larger than other scans.

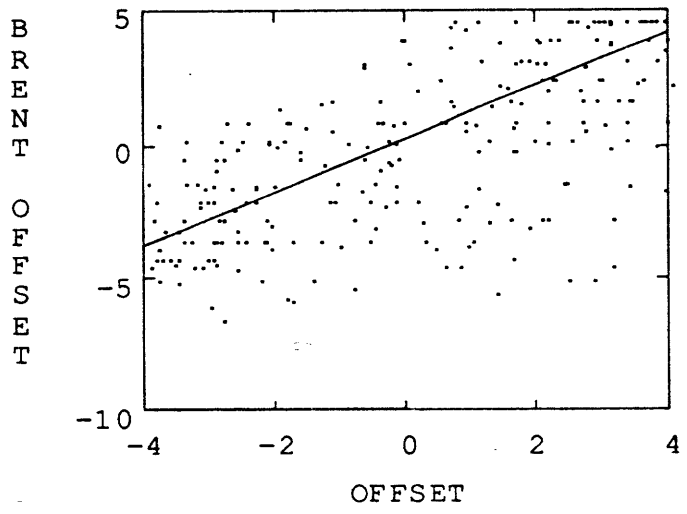

Figure 5.3b-1 Plot of estimated offset vs. the actual offset

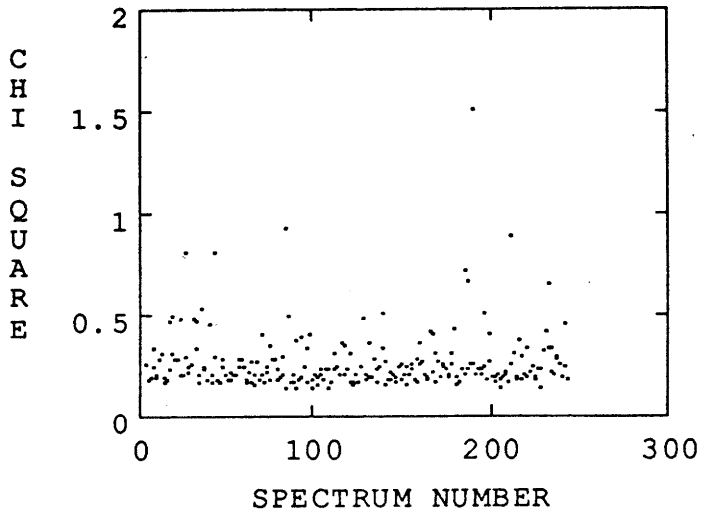

Figure 5.3b-2 Plot of Chi-square

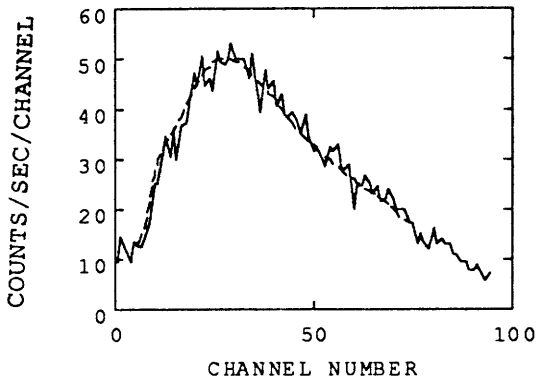

Figure 5.3b-3 The solid curve represents the 100th spectrum c(x) in sd36a and the dashed curve is the estimated spectrum for c(x). Chi-square is 0.1662.

Chapter 6 Conclusions

In this study, we developed the weighted least square error estimation and maximum likelihood estimation algorithms for gain and offset estimation of pulse height spectra. We tested the performance of the two estimation algorithms on two noise free scans and two noisy scans from one data ensemble

We did performance test on both least square error estimation and maximum likelihood estimation. The performance tests are done on two different scans and the noisy data generated from them. For the noise free scans, our estimation errors for gain are about two percent, and the standard deviations for offset are about 0.7 keV. With the standard gain at 3.2 keV/channel, a 0.7 keV offset is only one fifth of a channel. For the noisy scan generated corresponding to 600 ft/hr logging speed, our estimation errors for gain are about four percent, and the standard deviation for offset is about 1.7 keV. Only for the noisy scan generated corresponding to 1800 ft/hr logging speed, our estimations show big errors: about twelve percent for gain and 3 keV for offset.

The results from the performance tests show that our least square error estimation algorithm is good for gain and offset estimation.

References

[1] E. O. Brigham, *The Fast Fourier Transform*, Prince-Hall, Inc., 1974.

[2] C. H. Chen, *Issues in Acoustic Signal-Image Processing and Recognition*, Springer-Verlag Berlin Heidelberg, 1983.

[3] D. V. Ellis, *Well Logging for Earth Scientists*, Elsevier Science Publishing Co., Inc., 1987.

[4] M. P. Ekstrom, *Digital Image Processing techniques*, Academic Press, Inc., 1984.

[5] R. L. Heath, *Scintillation Spectrometry Gamma-Ray Spectrum Catalogue*, IDo-16880-1, AEC Research and Development Report, Physics, TID-450, 1964

[6] G. F. Knoll, *Radiation Detection and Measurement*, John Wiley & Sons, Inc., 1979.

[7] F. L. Lewis, *Optimal Estimation with an Introduction to stochastic Control Theory*, John Wiley & Sons, Inc., 1986

[8] National Council on Radiation protection and measurements, *Environmental Radiation Measurements*, NCRP Report No. 50,1977.

[9] A. Rosenfeld & A. C. Kak, *Digital Picture Processing*, Academic Press, Inc., 1976.

[10] W. H. Press, B. P. Flannery & W. T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Inc., 1988.

[11] C. C. Watson, "Optimal Linear Compression of X-Ray transmission Spectra", submitted to *IEEE Transaction on Acoustics, Speech, and Signal Processing*, 1989.

[12] C. C. Watson, "Optical Linear Compression of Noisy Data", submitted to *IEEE Transaction on Acoustics, Speech, and Signal Processing*, 1989.

[13] C. C. Watson, "The Compton and Photoelectric Spectroscopy of Multiply Scattered Photons", submitted to *IEEE Transaction on Nuclear Science*, 1990.

Appendix 1 Three Mapping Subroutines

```
/*****************************************************/
/*   Function  gainof()                              */
/*             Fourth order polinomial               */
/*             interpolation                         */
/*****************************************************/

      subroutine gainof (SP1,N1,OF1,GN1,SP2,N2,OF2,GN2)
c  SPECTRUM GAIN/OFFSET ADJUSTMENT ROUTINE


c  ARGUMENTS:
c    SP1,SP2  -  INPUT, OUTPUT SPECTRAL ARRAYS
c    N1,N2    -  UPPER CHANNEL LIMITS FOR SP1,SP2
c    OF1,OF2  -  OFFSETS OF SPECTRA (ENERGY UNITS)
c    GN1,GN2  -  GAINS OF SPECTRA (ENERGY/CHANNEL)
c
c  ENERGY VS. CHANNEL IS EXPRESSED BY:
c    ENERGY = GAIN * CHANNEL + OFFSET
c
c  THE BOTTOM OF THE FIRST CHANNEL IS DEFINED AS CHANNEL 0.
c
c  USES A 4TH-DEGREE LOCAL POLYNOMIAL TO EXPRESS THE UNDERLYING
c    COUNT-RATE VS. ENERGY FOR EACH CHANNEL IN THE UNADJUSTED
c    SPECTRUM
c
c  WRITTEN BY W. FRAWLEY, MODIFIED BY J. GRAU, ADAPTED BY AJB.
c
      real*8 SP1(1),SP2(1),XX(5,5),SUM,WW,POWTOP,POWBOT
      data XX / 0.1125, -1.45,  26.675, -1.45,  0.1125,
     1           1.25  , -8.5 ,   0.0  ,  8.5 , -1.25  ,
     2          -0.5   ,  6.0 , -11.0  ,  6.0 , -0.5   ,
     3          -0.5   ,  1.0 ,   0.0  , -1.0 ,  0.5   ,
     4           0.2   , -0.8 ,   1.2  , -0.8 ,  0.2   /
c
c  CHECK IF AN ADJUSTMENT IS NEEDED
      if (OF1.eq.OF2.and.GN1.eq.GN2) then
         do  I=1,MIN0(N1,N2)
            SP2(I) = SP1(I)
 end do
 go to 600
      end if
c  SET UP ADJUSTMENT PARAMETERS
```

```
      K1MAX = N1-5
      GN21 = GN2/GN1
      OF21 = (OF2-OF1-0.5*GN2)/GN1-2.0
      OF22 = (OF2-OF1)/GN1-2.5
C  ADJUST SPECTRUM
      do  K2=1,N2
 SUM = 0.0
 K1 = GN21*K2+OF21
 if (K1.ge.0.and.K1.le.K1MAX) then
            FTOP = GN21*K2-K1+OF22
            FBOT = FTOP-GN21
            POWTOP = 0.0416666667
            POWBOT = 0.0416666667
            do  J=1,5
               WW = 0.0
               do  K=1,5
                  WW = WW+XX(K,J)*SP1(K1+K)
               end do
               POWTOP = POWTOP*FTOP
               POWBOT = POWBOT*FBOT
               SUM = SUM+WW*POWTOP-WW*POWBOT
            end do
            SP2(K2) = SUM
 end if
      end do

 600  return
      end
```

```
/*****************************************************/
/*   Function  lgof()                                */
/*             linear interpolation                  */
/*****************************************************/

void lgof(c1, n1, of1, gn1, c2, n2, of2, gn2)
     int n1, *n2;
     double *c1, *c2;
     double  of1, gn1, of2, gn2;
{
  int i, j, k, l, h;
  double low, high, ly, hy;
  double a1, a2, b1, b2;
  double di, dj, x, y, fa, fb;

  a1 = gn1;
  b1 = of1;
```

```
  a2 = gn2;
  b2 = of2;
  fa = a2 / a1;
  fb = (b2 - b1)/a1;

  if (a1 == 0.0 || a2 == 0.0) {
    printf("gain cannot be zero!\n");
    exit(0);
  }

  low = (b1 - b2) / a2;
  high = ((double) n1)/fa + low;
  l = (int) low + 1;
  h = (int) high + 1;
  *n2 = h;

  for (i=0; i<h; i++) {
    c2[i] = 0.0;
  }

  if (l < 0)
    l = 0;

  if (a1 == a2 && b1 == b2) {
    for (i=0; i<h; i++)
      c2[i] = c1[i];
  }

  else {
    for (i=l; i<h; i++) {
      di = (double) i;
      y = di*fa + fb;
      j = (int) y;
      c2[i] = (c1[j+1] - c1[j]) * (y - (double) j) + c1[j];
      c2[i] = c2[i] * fa;
    }
  }
}


/*********************************************/
/*   Function  gofch()                       */
/*        Interpolation using Shannon's  */
/*             Sampling Theorem               */
/*********************************************/

void gofch(c1, n1, of1, gn1, c2, n2, of2, gn2)
     int n1, *n2;
```

```
     double *c1, *c2;
     double  of1, gn1, of2, gn2;
{
  int i, j, k, l, h;
  double low, high;
  double a1, a2, b1, b2;
  double di, dj, x, y, fa, fb;
  double pi = 3.1415926;

  a1 = gn1;
  b1 = of1;
  a2 = gn2;
  b2 = of2;
  fa = a2 / a1;
  fb = (b2 - b1)/a1;

  if (a1 == 0.0 || a2 == 0.0) {
    printf("gain cannot be zero!\n");
    exit(0);
  }

  low = (b1 - b2) / a2;
  high = ((double) n1)/fa + low;
  l = (int) low + 1;
  h = (int) high + 1;
  *n2 = h;

  for (i=0; i<h; i++) {
    c2[i] = 0.0;
  }

  if (a1 == a2 && b1 == b2) {
    for (i=0; i<n1; i++)
      c2[i] = c1[i];
  }

  else {
    for (i=0; i<h; i++) {
      c2[i] = 0.0;
      di = (double) i;

      for (j=0; j<n1; j++) {
dj = (double) j;
x = dj - di*fa - fb;

if (x < 1.0e-10 && x > -1.0e-10)
  y = 1.0;
else y = (sin(pi*x)) / (pi*x);
```

```
c2[i] += c1[j] * y;
      }
      c2[i] = c2[i] * fa;
    }
  }
}
```

Appendix 2 Subroutine File

Subrts.c

```
#include <stdio.h>
#include <math.h>
#include "/dsk/171/jhlu/s90/sd132/prin/lll.h"

int ncom=2;
double (*nrfunc)();
double pcom[2], xicom[2];

/*  UTILITY 1 ***********************************/
void nrerror(error_text)
     char error_text[];
{
  void exit();

  fprintf(stderr, "Numerical Recipes run-time error...\n");
  fprintf(stderr, "%s\n", error_text);
  fprintf(stderr, "...Now exiting to system...\n");
  exit(1);
}

/*  UTILITY 2 ***********************************************/
double *vector(nl, nh)
     int nl, nh;
{
  double *v;
  v = (double *) malloc((unsigned) (nh-nl+1)*sizeof(double));
  if (!v)
    nrerror("allocation failure in vector()");

  return (v);
}


/*  UTILITY 3 ***********************************************/
void free_vector(v, nl, nh)
     double *v;
     int nl, nh;
{
  free((char*) (v));
```

```
}

/*  UTILITY 4 ***************************************/

double dbabs(x)
     double x;
{
  double y;

  y = x;
  if (x < 0.0)
    y = -x;

  return (y);
}

/*************************************************************/
/* subrts.c  */

/*************************************************************/
/* Function 1.  Mult_matrices(a, b, c, arow, acol, bcol)   */
/*              a, b, c -- matrices                          */
/* Compute :    c = a * b                                    */
/*************************************************************/

void mult_matrices(a, b, c, arow, acol, bcol)
     double *a, *b, *c;
     int arow, acol, bcol;
{
  int i, j, k;
  double *temp;

  temp = c;

  for (i=0; i<arow; i++) {
    for (j=0; j<bcol; j++) {
      *temp = 0.0;
      for (k=0; k<acol; k++)
*temp += *(a + i*acol + k) * *(b + k*bcol + j);
      temp++;
    }
  }
}

/*****************************************************/
/* Function 2.     ulut(ut, lam, A, dim)             */
/*                 A   -- real symmetric matrix      */
/*                 ut  -- eigenvector matrix         */
```

```
/*                  lam -- eigenvalue array            */
/*    Compute :     A = transpose(ut) * lam * ut       */
/*******************************************************/

void ulut(ut, lam, A, dim)
     double *ut, *lam, *A;
     int dim;
{
  int i, j, k, dim2;
  double *u0;

  dim2 = dim * dim;
  u0 = (double *) malloc (dim2 * sizeof(double));

  printf("Initialize u0:\n");
  for (i=0; i<dim; i++) {
    for (j=0; j<dim; j++)
      *(u0 + i*dim + j) = *(ut + j*dim + i);
  }

  for (i=0; i<dim; i++) {
    for (j=0; j<dim; j++) {
      *(u0 + i*dim + j) *= *(lam + j);
    }
  }

  printf("Call mult_matrices:\n");

  mult_matrices(u0, ut, A, dim, dim, dim);
}

/*************************************************/
/*   Function 3. transpose(a, at, n1, n2)        */
/*     A(row, col): a(i, j) = at(j, i)           */
/*     n1: number of rows in a.                  */
/*     n2: number of cols in a.                  */
/*************************************************/

void transpose(a, at, n1, n2)
     double *a, *at;
     int n1, n2;
{
  int i, j, k;
  double *temp;

  temp = at;
  for (i=0; i<n2; i++) {
    for (j=0; j<n1; j++) {
```

```
      *temp = a[j*n2 + i];
      temp++;
    }
  }

}

/*************************************************/
/*   Function 4. invt(a, n)                      */
/*     a: general matrix                         */
/*************************************************/

void invt(A, n)
     double *A;
     int n;
{
  int i, j, k, job;
  int LDA;
  double round, *IPVT, *Z, DET[2], *WORK;

  IPVT = (double *) malloc (n * sizeof(double));
  Z = (double *) malloc (n * sizeof(double));
  WORK = (double *) malloc (n * sizeof(double));


  LDA = n;
  job = 1;
  dgeco_(A, &LDA, &n, IPVT, &round, Z);

  if (round + 1.0 == 1.0) {
    printf("round = %f\n", round);
    exit(0);
  }

  dgedi_(A, &LDA, &n, IPVT, DET, WORK, &job);

}


/*************************************************/
/*   Function 5. ma(B, a, c, n1, n2)             */
/*     B: matrix of n1 rows and n2 col's.        */
/*     a: array of dimension n2, represent for a */
/*        n2 x n2 diagonal matrix.               */
/*************************************************/

void ma(B, a, c, n1, n2)
     double *B, *a, *c;
```

```
     int n1, n2;
{
  double *temp;
  int i, j, k;

  temp = c;
  for (i=0; i<n1; i++) {
    for (j=0; j<n2; j++) {
      *temp = B[i*n2+j] * a[j];
      temp++;
    }
  }

}


/*********************************************/
/*   Function 6.  gofch()                    */
/*********************************************/

void gofch(c1, n1, of1, gn1, c2, n2, of2, gn2)
     int n1, *n2;
     double *c1, *c2;
     double  of1, gn1, of2, gn2;
{
  int i, j, k, l, h;
  double low, high;
  double a1, a2, b1, b2;
  double di, dj, x, y, fa, fb;
  double pi = 3.1415926;

  a1 = gn1;
  b1 = of1;
  a2 = gn2;
  b2 = of2;
  fa = a2 / a1;
  fb = (b2 - b1)/a1;

  if (a1 == 0.0 || a2 == 0.0) {
    printf("gain cannot be zero!\n");
    exit(0);
  }


  low = (b1 - b2) / a2;
  high = ((double) n1)/fa + low;

  l = (int) low + 1;
```

```
  h = (int) high + 1;
  *n2 = h;

  for (i=0; i<h; i++) {
    c2[i] = 0.0;
  }

  if (a1 == a2 && b1 == b2) {
    for (i=0; i<n1; i++)
      c2[i] = c1[i];
  }

  else {
    for (i=0; i<h; i++) {
      c2[i] = 0.0;
      di = (double) i;

      for (j=0; j<n1; j++) {
dj = (double) j;
x = dj - di*fa - fb;

if (x < 1.0e-10 && x > -1.0e-10)
  y = 1.0;
else y = (sin(pi*x)) / (pi*x);

c2[i] += c1[j] * y;
      }

      c2[i] = c2[i] * fa;
    }
  }

}

/*************************************************/
/*   Function 7.  lgof()                         */
/*                linear interpolation           */
/*************************************************/

void lgof(c1, n1, of1, gn1, c2, n2, of2, gn2)
     int n1, *n2;
     double *c1, *c2;
     double  of1, gn1, of2, gn2;
{
  int i, j, k, l, h;
  double low, high, ly, hy;
  double a1, a2, b1, b2;
  double di, dj, x, y, fa, fb;
```

```
    a1 = gn1;
    b1 = of1;
    a2 = gn2;
    b2 = of2;
    fa = a2 / a1;
    fb = (b2 - b1)/a1;

    if (a1 == 0.0 || a2 == 0.0) {
      printf("gain cannot be zero!\n");
      exit(0);
    }

    low = (b1 - b2) / a2;
    high = ((double) n1)/fa + low;

    l = (int) low + 1;
    h = (int) high + 1;
    *n2 = h;

    for (i=0; i<h; i++) {
      c2[i] = 0.0;
    }

    if (l < 0)
      l = 0;

    if (a1 == a2 && b1 == b2) {
      for (i=0; i<h; i++)
        c2[i] = c1[i];
    }

    else {
      for (i=l; i<h; i++) {
        di = (double) i;
        y = di*fa + fb;
        j = (int) y;
        c2[i] = (c1[j+1] - c1[j]) * (y - (double) j) + c1[j];
        c2[i] = c2[i] * fa;
      }
    }
}

/************************************************/
/*   Function 8.  extra()                       */
/************************************************/

void extra(c, d, ne)
```

```
     double *c;
     int d, ne;
{
  int i, j, n;
  int l, h;
  double x, y, dl, dh, k;

  l = d - 3;
  h = 80;

  dl = (double) l;
  dh = 80.0;
  y = c[l-1];
  k = y / (dh - dl);
  for (i=l; i<h; i++) {
    y = y - k;
    c[i] = y;
  }
}

/*************************************************/
/*   Function 9.  recon()                        */
/*************************************************/

void recon(c, ec, u, mc, n2, n1)
     double *c, *ec, *u, *mc;
     int n2, n1;
{
  int i, j;
  double *lam, *dc;

  lam = (double *) malloc (n1 * sizeof(double));
  dc = (double *) malloc (n2 * sizeof(double));

  for (i=0; i<n2; i++) {
    dc[i] = c[i] - mc[i];
    ec[i] = 0.0;
  }

  mult_matrices(u, dc, lam, n1, n2, 1);

  for (i=0; i<n1; i++) {
    for (j=0; j<n2; j++) {
      ec[j] += lam[i] * u[i*n2+j];
    }
  }

  for (i=0; i<n2; i++) {
```

```
    ec[i] += mc[i];
  }
}


/*************************************************/
/*   Function 10.  rms()                         */
/*************************************************/

void rms(c1, c2, n, er)
     double *c1, *c2, *er;
     int n;
{
  int i, j;
  double err, d, dn, x;

  dn = (double) n;
  err = 0.0;
  for (i=0; i<n; i++) {
    d = c1[i] - c2[i];
    x = c2[i];
    if (x <= 0.0) {
      printf("Singularity at rms  i = %d ; %f\n", i, c2[i]);
    }
    err += (d * d) / x;
  }
  *er = err/dn;
}

/*************************************************/
/*   Function 10-1.  rms1()                      */
/*************************************************/

void rms1(c1, c2, n, er)
     double *c1, *c2, *er;
     int n;
{
  int i, j;
  double err, d;

  err = 0.0;
  for (i=0; i<n; i++) {
    d = c1[i] - c2[i];
    err += d * d;
  }
  *er = err;
}
```

```
/*************************************************/
/*   Function 11.   fact(n)                      */
/*************************************************/

int fact(n)
     int n;
{
  int i, j;

  i = n;
  j = 1;
  if (n < 0) {
    printf("negative factorial, quit!\n");
    exit(0);
  }

  if (n == 0 || n == 1)
    j = 1;

  if (n>1) {
    for (i=n; i>0; i--)
      j = j * i;
  }
  return (j);
}


/*************************************************/
/*   Function 12. coeff(u, ut, h, mh, n1, n2, a) */
/*************************************************/

void coeff(u, ut, h, mh, n1, n2, a)
     double *u, *ut, *h, *mh, *a;
     int n1, n2;
{
  double *ph, *ph1, *b, *dh, *H1;
  int i, j, k;

  ph = (double *) malloc (n2 * n1 * sizeof(double));
  ph1 = (double *) malloc (n2 * n1 * sizeof(double));
  b = (double *) malloc (n1 * n1 * sizeof(double));
  dh = (double *) malloc (n2 * sizeof(double));
  H1 = (double *) malloc (n2 * sizeof(double));

  /* define: H1 = inverse of diagonal matrix H(x):  */
  for (i=0; i<n2; i++) {
    H1[i] = 1.0/mh[i];
```

```
    dh[i] = h[i] - mh[i];
  }

  /* Compute ph = u * H1:  */
  ma(u, H1, ph, n1, n2);

  /* compute b = ph * ut:  */
  mult_matrices(ph, ut, b, n1, n2, n1);

  /* compute inverse of b:  */
  invt(b, n1);

  /* compute ph = b * u:  */
  mult_matrices(b, u, ph, n1, n1, n2);

  /* compute ph1 = ph * H1:  */
  ma(ph, H1, ph1, n1, n2);

  /* compute the coefficient array a:  */
  mult_matrices(ph1, dh, a, n1, n2, 1);

}

/*************************************************/
/*   Function 13. lnfact(h, y)                   */
/*                Compute *y = ln(h!)             */
/*************************************************/

void lnfact(h, y)
     double h, *y;
{
  int n, m, i, j;
  double a, sum;

  n = (int) (h + 0.5);

  sum = 0.0;
  a = (double) n;

  if (n>1) {
    for (i=n; i>1; i--) {
      sum += log(a);
      a = a - 1.0;
    }
  }
  *y = sum;
}
```

```
/*****************************************************************/
/*   Function 14. sterling(h, y)                                 */
/*                Compute *y = ln(h!)                            */
/*                           = h ln(h) - h + .5ln(2 PI h)        */
/*****************************************************************/

void sterling(h, y)
     double h, *y;
{
  *y = h * log(h) - h + 0.5 * log(2.0*PI*h);
}


/*****************************************************/
/*   Function 15. compute_g(h, g, u, mh, n2, n1)  */
/*                Compute the estimated spectrum g*/
/*                g = ut * a + mh                  */
/*                a: coefficient of dimension n1   */
/*                n2: dimension of h, g, mh.       */
/*****************************************************/

void compute_g(h, g, u, mh, n2, n1)
     double *u, *h, *g, *mh;
     int n1, n2;
{
  int i, j;
  double *a, *ut;

  a = (double *) malloc (n1 * sizeof(double));
  ut = (double *) malloc (n1 * n2 * sizeof(double));

  /* compute ut = transpose(u)  */
  transpose(u, ut, n1, n2);

  /* compute coefficient a: */
  coeff(u, ut, h, mh, n1, n2, a);

  /* compute g = ut * a + mh:  */
  mult_matrices(ut, a, g, n2, n1, 1);
  for (i=0; i<n2; i++) {
    g[i] = g[i] + mh[i];
  }
}

/*********************************************************/
/*   Function 16. likelihood(h, g, n2, l)                */
/*                Compute the likelihood function        */
/*********************************************************/
```

```
void likelihood(h, g, n2, l)
     double *h, *g, *l;
     int n2;
{
  int i, j, fh, ih;
  double *lnh, p, sum;

  lnh = (double *) malloc (n2 * sizeof (double));

  /* compute lnh = ln(h)  */
  for (i=0; i<n2; i++) {
    lnfact(h[i], &p);
    lnh[i] = p;
  }

  /*  compute likelihood function p:  */
  sum = 0.0;
  for (i=0; i<n2; i++) {
    p = h[i] * log(g[i]) - g[i] - lnh[i];
    sum = sum + p;
  }
  *l = -sum;
}

/*************************************************************/
/*   Function 17. st_like(h, g, n2, l)                       */
/*                Compute the likelihood function            */
/*************************************************************/

void st_like(h, g, n2, l)
     double *h, *g, *l;
     int n2;
{
  int i, j, fh, ih;
  double *lnh, sum, p;

  lnh = (double *) malloc (n2 * sizeof (double));

  /* compute lnh = ln(h)  */
  for (i=0; i<n2; i++) {
    sterling(h[i], &p);
    lnh[i] = p;
  }

  /*  compute likelihood function p:  */
  sum = 0.0;
  for (i=0; i<n2; i++) {
```

```
    p = h[i] * log(g[i]) - g[i] - lnh[i];
    sum = sum + p;
  }
  *l = -sum;
}

/*********************************************************/
/*   Function 18. st_like1(h, g, n2, l)                  */
/*                Compute the likelihood function        */
/*********************************************************/

void st_like1(h, g, n2, l)
     double *h, *g, *l;
     int n2;
{
  int i, j, fh, ih;
  double x, y, sum, p;

  /* compute x = sum(sq(h-g)/g):  */
  rms(h, g, n2, &x);

  sum = 2.0 * PI;
  p = (double) n2;
  y = p * log(sum);

  sum = 0.0;
  for (i=0; i<n2; i++) {
    p = log(g[i]);
    sum = sum + p;
  }

  /*  compute likelihood function p:  */
  p = x + y + sum;
  *l = p/2.0;
}


/*****************************************************/
/*  Function 19.  lse(u, mc, c, g, d, p, pd, m)      */
/*  Compute Error:                                   */
/*          u: principal components matrix           */
/*          mc: mean vector                          */
/*          c: spectrum with unknown gain & offset   */
/*          g: reconstructed c                       */
/*          d: dimension of c                        */
/*          p: p[2], p[0] = of1, p[1] = gn1,         */
/*          pd: pd[0] = ns, the starting channel     */
/*              pd[1] = ne, the ending channel       */
```

```
/*          m: output least square error                */
/*******************************************************/

void lse(u, mc, c, g, d, p, pd, m)
     double *u, *mc, *c, *g, *p, *m;
     int d, *pd;
{
  int i, j, d2, ns, ne, dm;
  double *c2, *c3;
  double of1, gn1, err;

  c2 = (double *) malloc(DIMC * sizeof(double));
  c3 = (double *) malloc(DIMC * sizeof(double));

  of1 = p[0];
  gn1 = p[1];

  ns = pd[0];
  ne = pd[1];
  dm = ne - ns;

  /* Initialize gc to be zero: */
  for (i=0; i<DIMC; i++) {
    c2[i] = 0.0;
    c3[i] = 0.0;
    g[i] = 0.0;
  }

  /* use "gainof" Fortran subroutine to define c2 */
  gofch(c, d, of1, gn1, c2, &d2, OF, GN);

  if (d2 < ne) {
    extra(c2, d2, ne);
  }

  /* take the central part of c2:  */
  for (i=0; i<dm; i++) {
    c3[i] = c2[i+ns];
  }

  recon(c3, g, u, mc, dm, N1);
  rms(c3, g, dm, m);

}


/*******************************************************/
/*  Function 20.  lseg(u, mc, c, g, d, p, pd, m)    */
```

```
/*  Compute Error:                                          */
/*          u: principal components matrix                  */
/*          mc: mean vector                                 */
/*          c: spectrum with unknown gain & offset          */
/*          g: reconstructed c                              */
/*          d: dimension of c                               */
/*          p: p[2], p[0] = of1, p[1] = gn1,                */
/*          pd: pd[0] = ns, the starting channel            */
/*              pd[1] = ne, the ending channel              */
/*          m: output least square error                    */
/*******************************************************/

void lseg(u, mc, c, g, d, p, pd, m)
     double *u, *mc, *c, *g, *p, *m;
     int d, *pd;
{
  int i, j, d2, ns, ne, dm;
  double *c2, *c3;
  double of1, gn1;

  c2 = (double *) malloc(DIMC * sizeof(double));
  c3 = (double *) malloc(DIMC * sizeof(double));

  of1 = p[0];
  gn1 = p[1];

  ns = pd[0];
  ne = pd[1];
  dm = ne - ns;

  /* Initialize gc to be zero: */
  for (i=0; i<DIMC; i++) {
    c2[i] = 0.0;
    c3[i] = 0.0;
    g[i] = 0.0;
  }

 /* use "gainof" Fortran subroutine to define c2 */
  gofch(c, d, of1, gn1, c2, &d2, OF, GN);

  if (d2<ne) {
    extra(c2, d2, ne);
  }

  /* take the central part of c2:  */
  for (i=0; i<dm; i++) {
    c3[i] = c2[i+ns];
  }
```

```
  compute_g(c3, g, u, mc, dm, N1);
  rms(c3, g, dm, m);

}


/*****************************************************/
/*  Function 21    cost_func(args, nf, y)            */
/*****************************************************/

double cost_func(args, nf, y)
     double *y;
     int nf;
     struct arg_cost *args;
{
  double *u, *mc, *c, *g, *ofgn;
  int *pdm, d;

  u = args->uc;
  mc = args->mc;
  c = args->cg;
  g = args->g;
  d = args->dm;
  ofgn = args->ofgn;
  pdm = args->pdm;

  if (nf == 1) {
    lse(u, mc, c, g, d, ofgn, pdm, y);
  }

  if (nf == 2) {
    lseg(u, mc, c, g, d, ofgn, pdm, y);
  }
}


/*************************************************************/
/* Function 22  brent1(args, m, gf, ax, bx, cx, tol, xmin, y)  */
/*   args:   struct arg_cost, contains arguments needed (111.h) */
/*   m:      function chooser                                  */
/*           m=1, projection reconstruction                    */
/*           m=2, filtered   reconstruction                    */
/*   gf:     offset or gain                                    */
/*           gf=0, search for offset                           */
/*           gf=1, search for gain                             */
/*   ax, cx: upper and lower limit for minimum search          */
/*   bx:     start point for minimum search                    */
```

```
/*   tol:     tolerance                                              */
/*   xmin:    minimun occurs at this value                           */
/*   y:       cost function value                                    */
/***********************************************************************/

double brent1(args, m, gf, ax, bx, cx, tol, xmin, y)
     double ax, bx, cx, tol, *xmin, *y;
     int m, gf;
     struct arg_cost *args;

{
  int iter;
  double a, b, d, etemp, fu, fv, fw,fx,
  double p, q, r,tol1, tol2, u, v, w, x, xm;
  double e = 0.0;

  a = min(ax, bx);
  b = max(ax, bx);

  x = w = v = bx;

  *(args->ofgn + gf) = x;
  (*cost_func)(args, m, &fx);
  fw = fv = fx;

  for (iter=1; iter<=ITMAX2; iter++) {
    xm = 0.5 * (a + b);
    tol1 = tol * dbabs(x) + ZEPS;
    tol2 = 2.0 * tol1;

    if (dbabs(x-xm) <= (tol2 - 0.5*(b-a))) {
      *xmin = x;
      *y = fx;
      return (fx);
    }

    if (dbabs(e) > tol1) {
      r = (x - w) * (fx - fv);
      q = (x - v) * (fx - fw);
      p = (x - v) * q - (x -w) * r;
      q = 2.0 * (q - r);

      if (q > 0.0)
p = -p;

      q = dbabs(q);
      etemp = e;
      e = d;
```

```
      if (dbabs(p) >= dbabs(0.5*q*etemp) ||
          p <= q*(a-x) || p >= q*(b-x)) {
e = (x >= xm ? a-x : b-x);
d = CGOLD * e;
      }

      else {
d = p/q;
u = x+d;
if (u-a < tol2 || b-u < tol2)
  d = SIGN(tol1, xm-x);

      }
    }

    else {
      e = (x >= xm ? a-x : b-x);
      d = CGOLD * e;
    }

    u = (dbabs(d) >= tol1 ? x+d : x+SIGN(tol1, d));
    *(args->ofgn + gf) = u;
    (*cost_func)(args, m, &fu);

    if (fu <= fx) {
      if (u >= x)
a = x;
      else
b = x;

      SHFT(v, w, x, u);
      SHFT(fv, fw, fx, fu);
    }

    else {
      if (u< x)
a = u;
      else
b = u;

      if (fu <= fw || w == x) {
v = w;
w = u;
fv = fw;
fw = fu;
      }
```

```
      else if (fu <= fv || v == x || v == w) {
v = u;
fv = fu;
      }
    }
  }

  nrerror("Too Many iterations in BRENT");
  *xmin = x;
  *y = fx;
  return fx;
}
```